ioProgrammo Anno XI - N°8 (117)

ANTEPRIMA LINQ: ARRIVA IL LINGUAGGIO SQL INTEGRATO DIRETTAMENTE IN .NET. ORA PUOI TRATTARE I DATI COME OGGETTI E CLASSI

VERSIONE PLUS RIVISTA+LIBRO+CD €9,90 — VERSIONE STANDARD RIVISTA+CD €6,90

ioProgrammo

PER ESPERTI E PRINCIPIANTI

Poste Italiane S.p.A Spedizione in A.P. • D.L. 353/2003 (conv.in L.27/02/2004 n.46) art.1 comma 2 DCB ROMA Periodicità mensile • AGOSTO 2007 • ANNO XI, N.8 (117)

IL CODICE DELLA RETE

Crea il tuo web server personalizzato e scopri cosa si nasconde dietro il traffico dei dati su Internet

- **LA TEORIA:** quali sono i protocolli che muovono il web e come utilizzarli
- **LE BASI:** crea un'applicazione e mettila in attesa su una porta
- **LA PERSONALIZZAZIONE:** aggiungi un file di configurazione per adattare il server al tuo sistema
- **LO SCAMBIO DEI DATI:** trova la pagina nel disco locale e inviala correttamente al client

ESEMPI IN .NET

INTERNET
AJAX XML? NO GRAZIE
Utilizza JSON o addirittura il testo per scambiare i dati

DATABASE
USA SUBITO ORACLE BERKLEY DB
Crea software super performante con il DB integrato in Java

VISUAL BASIC
IL LOG DENTRO IL SOFTWARE
Fornisci al tuo programma un sistema di Logging per facilitare la ricerca di eventuali errori

PATTERN
ADDIO AL CODICE COMPLICATO
Con Facade "nascondi" la complessità delle classi e rendi tutto più semplice

C++ METTE D'ACCORDO LINUX E WINDOWS

Ecco come creare software che funziona su tutti i sistemi senza preoccuparsi delle differenze! Scopriamo le wxWidgets...

JAVASCRIPT
AJAX SÌ MA IN MODALITÀ PUSH
Appena il dato cambia sarà il server ad aggiornare la tua pagina

ASP.NET
ESTENDI IL LINGUAGGIO
Crea un super check box utilizzando la grafica bitmap

JAVA
NASCONDI BENE I TUOI SEGRETI
Crittografia dei dati. Usala subito con i nostri esempi

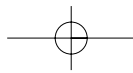
CORSI BASE
SMARTPHONE
La guida passo passo per realizzare la prima applicazione per Pocket PC

UML
Disegna il diagramma di flusso del tuo software, clicca e ottieni il codice!

JAVA SERVER PAGES
Gestisci la tua biblioteca con i managed Bean

ALGORITMI DI MOLTIPLICAZIONE sembra banale ma questa operazione di base influisce pesantemente sulle performance del sistema, ecco come ottimizzarla

IL CODICE DELLA RETE • C++ METTE D'ACCORDO LINUX E WINDOWS • AJAX IN MODALITÀ PUSH • ESTENDERE ASP.NET
EDIZIONI MASTER

2-07-2007 12:49:45

Anno XI - N.ro 08 (117) - Agosto 2007 - Periodicità: Mensile
Reg. Trib. di CS al n.ro 593 del 11 Febbraio 1997
Cod. ISSN 1128-594X
E-mail: ioprogrammo@edmaster.it
http://www.edmaster.it/ioprogrammo
http://www.ioprogrammo.it

Direttore Editoriale: Massimo Sesti
Direttore Responsabile: Massimo Sesti
Responsabile Editoriale: Gianmarco Bruni
Vice Publisher: Paolo Soldan
Redazione: Fabio Farnesi
Collaboratori: R. Allegra, D. De Michelis, F. Grimaldi, E. Viale, V. Vessia, A. Pelleriti, F. Smelzo, C Scuderi, G. Malaga, F. Fortino, V. Arconzo
Segreteria di Redazione: Rossana Scarcelli

Realizzazione grafica: Cromatika S.r.l.
Art Director: Paolo Cristiano
Responsabile grafico di progetto: Salvatore Vuono
Coordinamento tecnico: Giancarlo Sicilia
Illustrazioni: M. Veltri
Impaginazione elettronica: Francesco Cospite, Lisa Orrico, Nuccia Marra, Luigi Ferraro

Realizzazione Multimediale: SET S.r.l.
Realizzazione CD-Rom: Paolo Iacona

Pubblicità: Master Advertising s.r.l.
Via C. Correnti, 1 - 20123 Milano
Tel. 02 831212 - Fax 02 83121207
e-mail advertising@edmaster.it
Sales Director: Max Scortegagna
Segreteria Ufficio Vendite: Daisy Zonato

Editore: Edizioni Master S.p.a.
Sede di Milano: Via Ariberto, 24 - 20123 Milano
Sede di Rende: C.da Lecco, zona industriale - 87036 Rende (CS)
Presidente e Amministratore Delegato: Massimo Sesti
Direttore Generale: Massimo Rizzo

ABBONAMENTO E ARRETRATI
ITALIA: Abbonamento Annuale: IOPROGRAMMO (11 NUMERI) €59,90 SCONTO 21% SUL PREZZO DI COPERTINA DI €75,90 · IOPROGRAMMO CON LIBRO (11 NUMERI) €75,90 SCONTO 30% SUL PREZZO DI COPERTINA DI €108,90 OFFERTE VALIDE FINO AL 30/09/07.
Costo arretrati (a copia): il doppio del prezzo di copertina + € 5.32 spese (spedizione con corriere). Prima di inviare i pagamenti, verificare la disponibilità delle copie arretrate allo 02 831212.
La richiesta contenente i Vs. dati anagrafici e il nome della rivista, dovrà essere inviata via fax allo 02 83121206, oppure via posta a EDIZIONI MASTER via C. Correnti, 1 - 20123 Milano, dopo avere effettuato il pagamento, secondo le modalità di seguito elencate:

- cc/p n.16821878 o vaglia postale (inviando copia della ricevuta del versamento insieme alla richiesta);
- assegno bancario non trasferibile (da inviarsi in busta chiusa insieme alla richiesta);
- carta di credito, circuito Visa, Cartasì, o Eurocard/Mastercard (inviando la Vs. autorizzazione, il numero di carta di credito, la data di scadenza, l'intestatario della carta e il codice CVV2, cioè le ultime 3 cifre del codice numerico riportato sul retro della carta).
- bonifico bancario intestato a Edizioni Master S.p.A. c/o BCC MEDIOCRATI S.C.AR.L. c/c 0 000 000 12000 ABI 07062 CAB 80880 CIN P (inviando copia della distinta insieme alla richiesta).

SI PREGA DI UTILIZZARE IL MODULO RICHIESTA ABBONAMENTO POSTO NELLE PAGINE INTERNE DELLA RIVISTA. L'abbonamento verrà attivato sul primo numero utile, successivo alla data della richiesta.

Sostituzioni: qualora nei prodotti fossero rinvenuti difetti o imperfezioni che ne limitassero la fruizione da parte dell'utente, è prevista la sostituzione gratuita, previo invio del materiale difettato.
La sostituzione sarà effettuata se il problema sarà riscontrato e segnalato entro e non oltre 10 giorni dalla data effettiva di acquisto in edicola e nei punti vendita autorizzati, facendo fede il timbro postale di restituzione del materiale.
Inviare il CD-Rom difettoso in busta chiusa a:
Edizioni Master - Servizio Clienti - Via C. Correnti, 1 - 20123 Milano

Servizio Abbonati:
☎ **tel. 02 831212**
@ **e-mail: servizioabbonati@edmaster.it**

Assistenza tecnica: ioprogrammo@edmaster.it
Stampa: Arti Grafiche Boccia S.p.a. Via Tiberio Felice, 7 Salerno
Stampa CD-Rom: Neotek S.r.l. - C.da Imperatore - Bisignano (CS)
Distributore esclusivo per l'Italia: Parrini & C S.p.A.
Via Vitorchiano, 81 - Roma

Finito di stampare nel mese di Luglio 2007

Nessuna parte della rivista può essere in alcun modo riprodotta senza autorizzazione scritta della Edizioni Master. Manoscritti e foto originali, anche se non pubblicati, non si restituiscono. Edizioni Master non sarà in alcun caso responsabile per i danni diretti e/o indiretti derivanti dall'utilizzo dei programmi contenuti nel supporto multimediale allegato alla rivista e/o per eventuali anomalie degli stessi. Nessuna responsabilità è, inoltre, assunta dalla Edizioni Master per danni o altro derivanti da virus informatici non riconosciuti dagli antivirus ufficiali all'atto della masterizzazione del supporto. Nomi e marchi protetti sono citati senza indicare i relativi brevetti.

1 Anno di Computer Bild 2006, 1 Anno di Io Programmo in DVD 2006, 1 Anno di Linux Magazine in DVD 2006, 1 Anno di Office Magazine 2006, 1 Anno di Win Magazine in DVD 2006, Audio/Video/Foto Bild Italia, Calcio & Scommesse, Computer Bild Italia, Computer Games Gold, Digital Japan Magazine, Digital Music, DVD Magazine, DVD Magazine Films, Family DVD Games, Filmteca in DVD, Go!OnLine Internet Magazine, Home Entertainment, Horror Mania, I DVD di Quale Computer, I DVD di Win Magazine, I DVD de La Mia Barca, I Film di Idea Web, I Filmissimi in DVD, I Film di DVD Magazine, I Gadget de La Mia Barca, I Grandi Giochi per Pc, I Libri di Quale Computer, I Mitici all'Italiana, Idea Web, Idea Web Film, InDVD, IoProgrammo, I Tecnoplus di Win Magazine, Japan Cartoon, La mia Barca, La mia Videoteca, Le Femme Fatale del Cinema, Le Grandi Guide di Io Programmo, Linux Magazine, Miami Vice in DVD, Office Magazine, Play Generation, Play Generation Games, Popeye, PC Junior, Quale Computer, Softline Software World, Sport Life, Supercar in DVD, Star in DVD, Video Film Collection, Win Junior, Win Magazine Giochi, Win Magazine, Le Collection.

Questo mese su ioProgrammo

DOVE SONO GLI HACKER?

C'erano una volta gli hacker. Quelli buoni, quelli il cui nome non incuteva terrore, quelli che non erano associati a virus, spam e attentati alla sicurezza. Il loro spirito era del tutto diverso. Erano persone che "inventavano" ogni giorno un nuovo modo per approcciarsi all'informatica e in un certo senso anche alla vita. Si trattav a di trovare ogni giorno un nuovo limite, di oltrepassare le conoscenze già acquisite, alla caccia di altre nuove e più en usiasmanti. Esistono ancora queste persone? forse no. I programmatori sono impegnati a rispettare le scadenze per i propri clienti, a risolvere questo o quel problema legato alla produzione. Il tempo per studiare, per inventare è davvero limitato al minimo. Così capita addirittura che l'hack day sia sponsorizzato da una multinazionale come yahoo. Per carità, è bene che le multinazionali si occupino del popolo degli hacker, è bene che lo sforzo di tanti programmatori venga poi finalizzato per qualcosa di costruttivo. Tuttavia non possiamo non essere nostalgici di quel gruppo di persone che sviluppavano solo per il gusto di farlo. Una nota positiva vogliamo evidenziarla rispetto all'hack day di quest'anno ed è relativa alla presenza di tanti progetti dedicati all'informatica nel sociale. Tante idee rivolte ai portatori di handicap e comunque tese a migliorare la qualità della vita. Si tratta di un segnale importante, che denota come nel corso del tempo il messaggio che associa l'informatica al miglioramento della qualità della vita è diventato forte e importante. Ed i principali depositari di questa responsabilità siamo noi programmatori che con le nostre idee, le nostre capacità, il nostro spirito di innovazione ne siamo promotor. E' importante riservare del tempo per noi, al di la dei progetti commerciali a cui stiamo lavorando, in modo tale da non lasciare che quell'entusiasmo e quella curiosità che ci ha sempre distinto svanisca.

All'inizio di ogni articolo, troverete un simbolo che indicherà la presenza di codice e/o software allegato, che saranno presenti sia sul CD (nella posizione di sempre \soft\codice\ e \soft\tools\) sia sul Web, all'indirizzo http://cdrom.ioprogrammo.it.

IL CODICE DELLA RETE

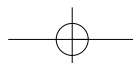
Crea il tuo Web server personalizzato e scopri cosa si nasconde dietro il traffico dei dati su Internet

LA TEORIA: Quali sono i protocolli che muovono il Web e come utilizzarli

LE BASI: crea un'applicazione e mettila in attesa su una porta

LO SCAMBIO DEI DATI: trova la pagina nel disco locale e inviala correttamente al client

LA PERSONALIZZAZIONE: aggiungi un file di configurazione per adattare il server al tuo sistema

pag. 14

004-005 editoriale 3-07-2007 9:54 Pagina 5

C++ METTE D'ACCORDO LINUX E WINDOWS

Ecco come creare software che funziona su tutti i sistemi senza preoccuparsi delle differenze! Scopriamo le wxWidgest...

pag. 88

IOPROGRAMMO WEB

Ajax visto con l'occhio del server . . . . **pag. 22**
Normalmente siamo abituati a pensare ad un meccanismo in cui un server risponde alle domande di un client. E se avvertissimo i ruoli? Ecco come fare

Utilizziamo Ajax senza XML . . . . **pag. 30**
XML può essere una soluzione comoda per molti problemi, ma in qualche caso potrebbe essere utile lavorare con altri tipi di formati, ad esempio Json o testo. Vediamo come svincolare Ajax dalla schiavitù di XML

SISTEMA

Creiamo un Asp.Net personalizzato

pag. 38

Una delle maggiori possibilità offerte da Asp.Net è quella di poter essere esteso sulla base delle esigenze dei programmatori

SISTEMA

Metti al riparo i tuoi dati con Java . . . . **pag. 44**
Proteggere i propri dati contro occhi indiscreti, ai nostri giorni assume una rilevanza senza precedenti. Java mette a disposizione meccanismi semplici per la sicurezza. ecco le basi per imparare a crittografare i dati

Mettiamo insieme Skipe e google . . . . **pag. 53**
Integriamo due tra le più diffuse tecnologie del momento: Google maps e Skype. Realizziamo una semplice applicazione per visualizzare la lista contatti di Skype sulle mappe messe a disposizione dal colosso californiano

DATA BASE

DAta BAse Berkley integrato in Java . . . . **pag. 56**
Una panoramica su uno dei DB più performanti del momento. Può essere utilizzato in modo embedded ed è stato realizzato appositamente per il linguaggio di Sun. Impariamo come sfruttarlo al massimo

Linq il linguaggio SQL dentro .Net . . . . **pag. 62**
Linq: il sistema di interrogazione dei dati rappresenta forse uno dei componenti più importanti introdotti dal .Net Framework 3.5, Impariamo ad utilizzarlo affrontando anche le altre novità del nuovo framework

VISUAL BASIC

Un sistema di Log sofisticato in VB6 . . . . **pag. 68**
Fornire un sistema di log basato su file alle applicazioni è cosa banale , ma rileggere, classificare e gestire le informazioni tracciate è tutt'altro che semplice vediamo come scriverle e rileggerle da DB, XML e File di testo

PATTERN

Facade: un sistema a colpo d'occhio . . . . **pag. 81**
Proprio come facciamo con i singoli oggetti, a volte vogliamo nascondere la complessità di un intero sistema dietro un'interfaccia semplice. Farlo non è difficile, basta applicare il pattern facade. Ecco come...

CORSI BASE

Modelliamo il software con UML **pag.84**

RUBRICHE

Gli allegati di ioProgrammo pag. 8
Il software in allegato alla rivista

Il libro di ioProgrammo pag. 6
Il contenuto del libro in allegato alla rivista

News pag. 12
Le più importanti novità del mondo della programmazione

Tips & Tricks pag. 76
Una raccolta di trucchi da tenere a portata di... mouse

Software pag. 109
I contenuti del CD allegato ad ioProgrammo.

Utilizziamo UML per progettare un software per la gestione della nostra squadra di calcio. Team2007 è un'applicazione che non può mancare nel computer di ogni presidente attento alla tecnologia

Programmare gli Smartphone . . . . **pag. 98**
Avete mai pensato di personalizzare PDA o cellulare con applicazioni create da voi? VB.NET consente di farlo piuttosto semplicemente minimizzando i tempi di sviluppo e facilitando il rilascio di programmi per Smart Device...

Usiamo JSF con Managed Bean . . . . **pag. 103**
Impariamo come si gestiscono i dati tramite le form, Come è possibile validarli e come stampare a schermo le informazioni fornite in input. vedremo come il ciclo di vita di un'applicazione JSF è decisamente diverso da quello classico

GRAFICA

C++ scopre le finestre! . . . . **pag. 102**
Come si crea una finestra in c++? Una semplice domanda apre un universo di risposte e di librerie. Scopriamo quali sono e cominciamo a sviluppare interfacce grafiche superformanti e multipiattaforma

SOLUZIONI

Algoritmi di moltiplicazione . . . . **pag. 110**
La fondamentale operazione della moltiplicazione può essere compiuta in molti modi, alcuni erano già noti alle antiche civiltà come quella egizia. vediamo quali sono i metodi che velocizzano la CPU

QUALCHE CONSIGLIO UTILE

I nostri articoli si sforzano di essere comprensibili a tutti coloro che ci seguono. Nel caso in cui abbiate difficoltà nel comprendere esattamente il senso di una spiegazione tecnica, è utile aprire il codice allegato all'articolo e seguire passo passo quanto viene spiegato tenendo d'occhio l'intero progetto.

http://forum.ioprogrammo.it

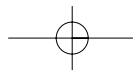
006 Presentaz Libro 2-07-2007 17:07 Pagina 6

I contenuti del libro

Windows presentation foundation

Oggi tutto è vettoriale, tutto eccetto la vecchia grafica. Windows fino ad ora si è basata sulla grafica BMP, pesante e non così maneggevole come quella vettoriale. E' ora di cambiare, e per questo è nata Windows Presentation Foundation: la tecnologia profondamente legata a Windows Vista e integrata con il .NET framework 3.0. Windows Presentation Foundation sostituisce la vecchia grafica Bitmap, aggiunge inoltre nuove funzioni che consentono un evoluzione verso il 3D. Il vettoriale ben si presta a rotazioni, ridimensionamenti, rapidi cambi di forma che non potrebbero altrimenti essere possibili con il BMP. Inoltre esistono una serie di estensioni che lo rendono fruibile anche dai browser e dalle applicazioni Internet.
Microsoft ha investito molto in questa tecnologia. Con questo HandBook di consultazione rapida vengono fornite tutte le indicazioni per lavorare subito con il nuovo Windows Presentation Foundation e fare evolvere le vostre applicazioni verso il magico mondo delle interfacce grafiche vettoriali

PROGRAMMA SUBITO CON LA TECNOLOGIA ALLA BASE DELLE NUOVE INTERFACCE GRAFICHE

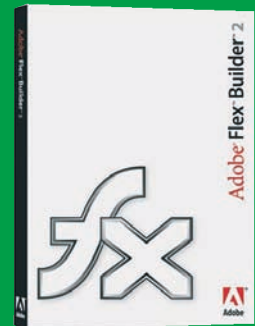
- **Introduzione ad XAML**
- **Sviluppare applicazioni con WPF**
- **User & Custom Controls**
- **Audio Video e Documenti**

Le versioni di ioProgrammo

Versione BASE

RIVISTA + CD-ROM in edicola

ADOBE FLEX BUILDER 2.0

Flash con qualcosa in più

Dove sono i limiti di flash? da un punto di vista di un programmatore il problema è una scarsa interazione con i database, un linguaggio che comunque diverge dalle tecniche classiche a cui siamo abituati. Ed ecco che arriva Flex. Meno grafica, meno ambienti dedicati ai designer, più programmazione, più interazione con i database, più strutture complesse, più XML. E' proprio XML la chiave di volta di Flex. Si crea un file XML che rappresenta la struttura di un documento lo si da in pasto al compilatore e quello tira fuori un file swf. A fare da contorno a tutto ciò c'è eclipse. Infatti per agevolare l'uso dell'SDK, Adobe ha sviluppato un plugin per il noto eclipse. L'accoppiata è di quelle vincenti. Il solo SDK di Flex è gratuito, il prodotto completo integrato in eclipse ha invece un valore commerciale di 449 € più IVA

Come usare l'interfaccia del CDRom

IL SOFTWARE
Una accurata recensione dei contenuti

IN EVIDENZA
Il top software del mese individuato dalla redazione

IL SOFTWARE
Il software diviso in categorie per una comoda consultazione

HOME
Torna alla pagina iniziale del CD-ROM

CONTATTACI
Vuoi inviare una email alla redazione con le tue richieste

TOP SITES
I siti più interessanti del mese selezionate per te

RICERCA SOFTWARE
Il database di tutti i software pubblicati da ioProgrammo anche gli arretrati

IL SOFTWARE
L'elenco del software contenuto nelle categorie

DIMENSIONE
La dimensione del software sul CD

SALVA
Clicca qui per installare o salvare il software sul tuo PC

INFO
Abbonamenti informazioni e servizi utili

Il software di ioProgrammo - n. 116 - Luglio 2007

www.ioprogrammo.it
ioPROGRAMMO CD-ROM
Anno XI - n.7 (116) - Luglio 2007
tiscali.

In evidenza
Tools
Indispensabili
Codice
Richiesta Software
Microsoft Webcast

Struts 2.0.6
Thinwire 1.2

Borland Jbuilder 2005 Foundation
L'eccezionale ambiente di Borland per lo sviluppo Java

Jbuilder è l'ambiente proposto da Borland per lo sviluppo di applicazioni Java. Come è tradizione di Borland si tratta di un ambiente incredibilmente curato nei particolari. Dotato di tutte le caratteristiche essenziali per un ottimo editor Java, la versione Foundation ha dalla sua parte il costo praticamente nullo, dovuto alla sua licenza, l'uso della foundation è infatti gratuito per usi personali. Il software è completo e senza scadenza.

Nel CD: JbuilderFoundation
Dimensione: 81,0 MB

Salva

EDIZIONI MASTER

HOME | CONTATTACI | TOP SITES | RICERCA SOFTWARE | INFO | ESCI

010 2-07-2007 16:10 Pagina 10

GLI ALLEGATI DI IOPROGRAMMO

Questo mese tre webcast dedicati ad estendere le funzionalità di ASP.NET

ASP.NET 2.0 AJAX: Extending ASP.NET AJAX (Livello 200)

Pensate che quanto offerto dalle ASP.NET 2.0 AJAX Extensions e dall'ASP.NET AJAX Control Toolkit sia insufficiente per le vostre esigenze? Volete creare un vostro componente o un vostro controllo? Questo webcast potrebbe fare al caso vostro: seguiteci.
Speaker Davide Vernole

ASP.NET 2.0 AJAX: Localization and Globalization (Livello 200)

Le nostre applicazioni non posso più fare a meno di includere il supporto per lingue diverse. In questo webcast esploreremo come sia possibile abilitare le nostre applicazioni ASP.NET, basate sulle estensioni di AJAX, al supporto multilingua.
Speaker Davide Vernole.

Virtual Earth API's - integrazione di mappe in siti Web (base)

Vuoi offrire delle informazioni correlate alla posizione geografica di luoghi o servizi? In questo webcast: concetti base per inserire e gestire mappe di Virtual Earth in pagine Web.
Speaker Mauro Rizzi.

PERCORSI

Telefonare e inviare SMS utilizzando TAPI (Livello 200) • Sviluppo per dispositivi mobili • Gestione della batteria tramite notifiche in Windows Mobile (Livello 200) • Scrivere applicazioni che utilizzano Mobile GPS (Livello 200) • Creazione di una chat con Bluetooth (Livello 200) • Costruzione di un servizio in un dispositivo (Livello 200) • Analisi e gestione dei log in IIS: casi pratici di utilizzo di LogParser (Livello 200) • I vantaggi del software originale • Il servizio Innovazione & Finanziamenti: Scopri come sviluppare con i fondi pubblici • per maggior informazioni visita: http://www.microsoft.com/italy/msdn/risorsemsdn/eventi/webcast/default.mspx

FAQ

Cosa sono i Webcast MSDN?
MSDN propone agli sviluppatori una serie di eventi gratuiti online e interattivi che approfondiscono le principali tematiche relative allo sviluppo di applicazioni su tecnologia Microsoft. Questa serie di "corsi" sono noti con il nome di Webcast MSDN

Come è composto tipicamente un Webcast?
Normalmente vengono illustrate una serie di Slide commentate da un relatore. A supporto di queste presentazioni vengono inserite delle Demo in presa diretta che mostrano dal vivo come usare gli strumenti oggetto del Webcast

Come mai trovo riferimenti a chat o a strumenti che non ho disponibili nei Webcast allegati alla rivista?
La natura dei Webcast è quella di essere seguiti OnLine in tempo reale. Durante queste presentazioni in diretta vengono utilizzati strumenti molto simili a quelli della formazione a distanza. In questa ottica è possibile porre domande in presa diretta al relatore oppure partecipare a sondaggi etc. I Webcast riprodotti nel CD di ioProgrammo, pur non perdendo nessun contenuto informativo, per la natura asincrona del supporto non possono godere dell'interazione diretta con il relatore.

Come mai trovo i Webcast su ioProgrammo
Come sempre ioProgrammo cerca di fornire un servizio ai programmatori italiani. Abbiamo pensato che poter usufruire dei Webcast MSDN direttamente da CD rappresentasse un ottimo modo di formarsi comodamente a casa e nei tempi desiderati. Lo scopo tanto di ioProgrammo, quanto di Microsoft è infatti quello di supportare la comunità dei programmatori italiani con tutti gli strumenti possibili.

Su ioProgrammo troverò tutti Webcast di Microsoft?
Ne troverai sicuramente una buona parte, tuttavia per loro natura i webcast di Microsoft vengono diffusi anche OnLine e possono essere seguiti previa iscrizione. L'indirizzo per saperne di più è: www.microsoft.it/msdn/webcast. segnalo nei tuoi bookmark. Non puoi mancare.

L'iniziativa sarà ripetuta sui prossimi numeri?
Sicuramente si.

News

DEBUTTO UFFICIALE PER LA GPL3

Il 29 Giugno la versione 3 della famosa licenza GPL che tutela i diritti del Free Software ha finalmente visto la luce. Per molti potrebbe non voler dire assolutamente niente, ma per le vagonate di software OpenSource attualmente presenti sul mercato e per le implicazioni che questa licenza ha persino su certi accordi commerciali di grande rilevanza, il rilascio della nuove versione assume un'importanza senza precedenti. La GPL3 vede la luce sotto i rflettori delle polemiche, non sono pochi gli attacchi che infatti sono stati portati a Stallman e al suo gruppo a causa delle modifiche apportate alla nuova licenza. In molti addirittura non ne vedevano la necessità. Eppure le novità se pur non immediatamente comprensibili per un pubblico di non addetti ai lavori riguardano soprattutto l'uso del DRM e tendono in tutti i modi a scoraggiarne l'utilizzo. Sono stati resi più restrittivi i diritti che proteggono la distribuzione del software Free. Di fatto con la nuova licenza nessuna azienda che abbia inserito del codice in software protetto da GPL3 potrà più reclamarne i diritti in un momento successivo.
Free Software Foundation ha annunciato che già 15 programmi del progetto GNU saranno convertiti alla nuova licenza, il primo è più noto è il pacchetto di compressione tar.

NUOVI TOOL PER IPHONE

È appena arrivato sul mercato e già si è scatenata la corsa per il rilascio di tool di sviluppo e compilatori. La nuova meraviglia tecnologica di casa Apple si configura come terreno fertile per gli sviluppatori. Così insieme a Google con il suo Google GWT e Adobe con il suo Apollo è adesso il turno di Morfik. L'SDK si preannuncia come un'insieme di applicazioni per il Web 2.0 e ottimizzate per l'iPhone. Staremo a vedere se i nuovi strumenti di sviluppo e la piattaforma di iPhone riusciranno a creare nuovi spazi per i programmatori.

REPORTAGE DALL'HACKDAY

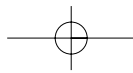
Il 16 e 17 giugno si è tenuto all'Alexandra Palace, un enorme centro esposizioni vittoriano di Londra, il primo Hack Day europeo. L'evento è stato sponsorizzato da Yahoo! e dalla BBC e ha riunito circa 290 hacker venuti specialmente dall'Inghilterra, ma anche dalla Germania, dalla Grecia, dalla Spagna, dalla Francia, dalla Danimarca e dagli Stati Uniti, oltre a cinque italiani. L'evento tra le altre cose mirava a dimostrare che per Hacker non si intende colui che penetra nei siti per diffondere virus o per leggere dati, ma al contrario, come ha enfaticamente detto Chad Dickerson, Sr. Director della Yahoo! Developer Network: il termine hacker "è positivo e creativo, anzi, estetico. L'hacker è un artista". Infatti l'hacker è un costruttore di software quando scrive o sviluppa un codice, un costruttore di hardware quando tratta i meccanismi elettronici e non, o entrambi, nel caso del montaggio e la programmazione di un robot. Lo scopo di riunire questi creatori, precedentemente selezionati da Yahoo!, era creare dei progetti originali usando le API della BBC e di Yahoo stessa usando l'insieme delle interfacce di programmazione come Programme Information, Weather Information, Flickr, del.icio.us e così via.
La mattina del sabato è stata dedicata alle conferenze per far familiarizzare i partecipanti con alcuni aspetti delle API, che sono liberamente disponibili su Internet. Al termine dell'evento i progetti sono stati esposti in presentazioni lampo di 60 secondi l'una e una giuria ha selezionato i 15 team finalisti e il progetto giudicato migliore. Si sono visti incroci di dati e integrazione tra foto, video, contenuti, musica, ecc. per i fini più disparati mentre il progetto che ha vinto è stato l'unico ad utilizzare RFID per sincronizzare cellulare e computer con dati e informazioni diverse a seconda dei luoghi e dei momenti della giornata (http://nytlabs.com/shifd/).
Per due giorni i partecipanti sono stati in una specie di isolamento ascetico, 280 ragazzi intorno ai 25 anni e una decina di donne, in questo enorme palazzo lontano da tutto, immerso nel verde e sovrastante la città. Gli hacker erano concentratissimi a programmare, a fare calcoli, a montare, incollare, smontare e assemblare, in un'apparente e comoda libertà. Ma i PR e il "servizio d'ordine" di Yahoo! e BBC erano ben presenti e circolavano discretamente fra i tavoli. Tutti i cliché classici del geek sono stati rispettati: la sala era enorme e buia; la mattina quasi non si poteva entrare a causa del fumo che saliva dal palco dove parlavano i primi speaker che spiegavano l'uso delle API, creando un'atmosfera da concerto rock; sotto ogni tavolo c'era una ciabatta per le prese chilometrica; avevano tutti una scheda per connettersi con password rigorosamente segreta e in fondo all'enorme sala c'era un tavolo pieno di junk food e sempre ben rifornito.
Nonostante l'intento dichiarato degli organizzatori fosse quello di creare un ambiente adatto e collaborativo per la creazione di progetti -- per avere un'idea dei gusti e i bisogni dell'elite di Internet e, quindi, per conoscere gli orientamenti di mercato – solo i team che già si conoscevano lavoravano insieme e chiacchieravano, condividendo i pochi momenti di inaspettato fuori-programma come un temporale, che ha costretto tutti a stare nel patio, o la connessione che andava a singhiozzi. Nell'era di Web 2.0, dove ci sono ancora spazi, l'idea generale era quella di farsi largo nella giungla di Internet, di entrare a lavorare in Yahoo! o nella BBC, perché i finalisti avrebbero avuto le interviste di lavoro con queste aziende, o di farsi comunque conoscere. Allo stesso tempo, molti parlavano di stare in modo diverso coi compagni di lavoro, rispettare l'ambiente, pensare al sud del mondo che sta peggio che in Occidente, rispettare il pianeta e l'atmosfera e così via. Gli hacker non sono solo dei costruttori ma stanno a metà strada fra i cowboy alla ricerca dell'oro e i nuovi hippy del nuovo millennio.

Enrica Garzilli

25.000 EURO PER L'IMAGINE CUP

Giunta alla sua quarta edizione, l'Imagine Cup è la manifestazione che ogni anno premia un giovane che si distingue nel campo della tecnologia. La competizione segue i ritmi di un normale torneo che con dei testa a testa serrati porta infine i gruppi migliori alla finale. Il vincitore intascherà un premio di 25.000 Euro, ma soprattutto ha la possibilità di mettere in luce le proprie capacità agli occhi del patron Bill Gates. La classifica dei migliori al momento in cui scriviamo è guidata da un gruppo di Indiani, a seguire abbiamo Polonia, Romania e Cina. A ridosso delle prime posizioni si contendono la piazza d'onore un nutrito gruppo proveniente ancora dalla Cina interrotto qua e la da qualche progetto Indiano. Se non ci fossero i paesi dell'Est europeo a tenere alto l'onore della vecchia Europa dovremmo scendere intorno al cinquantesimo posto per trovare un Francese. Completamente dispersa l'Italia. Non ci sono tracce neanche della Germania, dell'inghilterra, della Svizzera e persino l'America non ha una grande rappresentanza. A quanto pare l'informatica non parla la lingua dei paesi più industrializzati. I progetti in concorso sono di varia natura, c'è da registrare un calo della programmazione di videogiochi che invece fino all'anno scorso l'aveva fatta da padrone ed un notevole innalzamento della presenza di progetti dedicati all'innalzamento della qualità della vita ad esempio dei diversamente abili. Non mancano proposte decisamente innovative che vedono l'utilizzo di dispositivi completamente nuovi che fanno uso di touchscreen e tecnologie ancora non eccessivamente diffuse a livello commerciale. Il concorso è davvero interessante, peccato l'assenza di giovani programmatori Italiani, ma confidiamo nel prossimo anno.

JUST SAY NO TO GOOGLE

Quale programmatore non ha mai sognato di andare a lavorare in Google? Eppure c'è anche chi sostiene che il lavoro in Google non è tutto rose e fiori. A dirlo è un Ex dipendente della società che sviluppa e gestisce il potente motore di ricerca. Questo anonimo, tornato tra le braccia di mamma Microsoft dopo un'esperienza in Google ne racconta pregi e difetti in un lucido blog.
Secondo questo ben informato, Google tende ad abbracciare i propri dipendenti in modo totalitario impedendogli progressivamente di avere una vita privata al di fuori dell'ambiente lavorativo. Questa totale e progressiva astrazione del lavoratore dal mondo reale non sarebbe dovuta altro che alla grande quantità di vantaggi esclusivi che l'ambiente lavorativo mette a disposizione dei propri dipendenti. Si parla di una sorta di paese dei balocchi per ingegneri in cui cibo, relax, divertimenti, cure mediche specialistiche, servizi di trasporto privati ed un quantitativo straordinario di altri benefit vengono riuniti all'interno della posizione aziendale. Attratti dalla qualità della vita offerta dall'azienda, i giovani programmatori finiscono per dedicare all'azienda tutto il proprio tempo, trasformando il posto di lavoro in una sorta di abitazione, e vincolando la propria vita necessariamente a quella del proprio lavoro.
L'anonimo in questione tuttavia si sente di consigliare a Microsoft di adottare alcune strategie proposte proprio da Google. Si parla dei Tech Stop, ovvero una sorta di "negozio-cash & carry" con assistenza incorporata, dove i dipendenti possono cambiare il proprio computer in 5 minuti netti, ottenere una nuova scheda grafica e tutto ciò che serve al normale svolgimento del lavoro senza dovere affrontare una noiosa trafila burocratica. Allo stesso modo si parla di fornire cibo gratis ai dipendenti e di tutta un'altra serie di consigli presi proprio in prestito dall'esperienza in Google. Insomma alla fine, non sembra proprio che l'anonimo in questione sia riuscito a sottrarsi all'abbraccio "mortale" della cultura aziendale di Google.

DEVELOPER CI SEI? SEI CONNESSO?

AL GIORNO D'OGGI TUTTO È COLLEGATO, TUTTO VIAGGIA IN RETE. MA I TUOI PROGRAMMI SFRUTTANO ABBASTANZA LA POTENZA DEL NETWORKING? SVILUPPIAMO UN WEB SERVER FATTO IN CASA ED IMPARIAMO COSA FA MUOVERE REALMENTE INTERNET

Ogni volta che leggiamo la posta, navighiamo su un sito web, ci colleghiamo ad un sito ftp o utilizziamo programmi di file-sharing o di messaggistica, stiamo utilizzando un protocollo di rete. Un protocollo è un insieme di regole per l'invio e la ricezione di informazioni su dei canali di comunicazione. Tutti i protocolli di comunicazione fanno riferimento al modello ISO/OSI che definisce un'architettura stratificata formata da sette livelli.

1. **Fisico**: si occupa della trasmissione dei dati.
2. **Dati**: si occupa della rilevazione degli errori e della corretta trasmissione.
3. **Rete**: si occupa di stabilire e mantenere la comunicazione; controlla aspetti come il routing e la congestione della rete.
4. **Trasporto**: si occupa del trasferimento affidabile dei dati tra due host.
5. **Sessione**: si occupa delle comunicazioni tra applicazioni.
6. **Presentazione**: si occupa della conversione dei dati tra diversi formati per renderli compatibili con la piattaforma di destinazione.
7. **Applicazione**: è lo strato dedicato ai servizi per gli utenti, fornisce dei protocolli che interagiscono con le applicazioni.

Quando parliamo di architetture di rete, pensiamo alla suite di protocolli chiamata TCP/IP. Questa architettura è confrontabile con il modello ISO/OSI anche se non implementa alcuni strati.

Fig. 1: Il modello TCP/IP a confronto con l'architettura ISO/OSI

1. **Host to Network**: per l'invio dei pacchetti in rete.
2. **Internet (IP)**: permette ad un host di inviare pacchetti e farli viaggiare indipendentemente gli uni dagli altri.
3. **Trasporto (TCP/UDP)**: per consentire la comunicazione tra due endpoint. Abbiamo due protocolli: TCP (Transmission Control Protocol) che è connection oreiented e UDP (User Datagram Protocol) non orientato alla connessione.
4. **Applicazione**: contiene i protocolli di alto livello usati dalle applicazioni: esempi sono *Telnet*, *FTP* (*File Transfer Protocol*), *SMTP* (*Simple Mail Transfer Protocol*), *POP* (*Post Office Protocol*) o *HTTP* (*HyperText Transfer Protocol*).

Il Framework .Net contiene numerose classi per la gestione dei protocolli di rete in diversi namespace tra cui **System.Net, System.Net.Mail, System.Net.NetworkInformation** o **System.Net. Sockets**.

SOCKET SU TCP/IP

Come primo esempio implementiamo un software che utilizza i *Socket* per la comunicazione tra due *endpoint*. Un **Socket** è un canale di comunicazione che consente di inviare e ricevere dei dati attraverso la rete utilizzando un protocollo di comunicazione. Il Framework consente di gestire Socket su *TCP* e *UDP* tramite le classi **TcpClient, TcpListener,** e **UdpClient**.
Per provare le potenzialità del Framework, realizziamo un semplice web server capace di gestire richieste a pagine html statiche, senza supporto per lo scripting.
Utilizziamo la classe **TcpListener**, la quale fornisce dei metodi per stare in ascolto su una porta

014-020 29-06-2007 17:27 Pagina 15

in attesa di richieste di connessione e per gestirle in modalità asincrona.

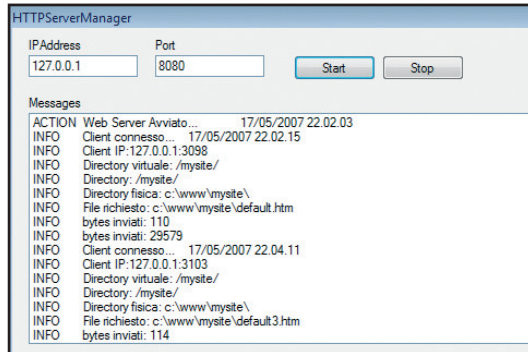



Fig. 2: Il nostro server web in azione!

Fig. 3: Il file ServerSetting.settings contenente le impostazioni del server

Creiamo il progetto "HTTPServer" ed aggiungiamo:

- una **MainForm** che conterrà le funzioni di avvio/interruzione servizio e una ListBox per i messaggi
- un file **ServerSettings.settings** che conterrà le impostazioni del server
- una classe **HTTPConfigReader.vb** per la lettura del file di configurazione
- una classe **HTTPServerManager.vb** che rappresenta il kernel del webserver

Il file di configurazione contiene tre variabili di tipo **StringCollection** (Namespace **System. Collections. Specialized**).

Vediamole:

DefaultFiles: è la lista dei possibili file di default

```
default.htm
default.html
```

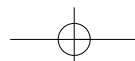
Directories: le coppie directory virtuale / directory fisica separate da uno spazio

```
/ C:\www/
/mysite/ C:\www\mysite\
```

MimeTypes: i tipi mime conosciuti (coppie estensione / tipo mime)

```
.html text/html
.htm text/html
.bmp image/bmp
.gif image/gif
.jpg image/jpg
```

Una volta salvato il file di configurazione passiamo al reader. Questo contiene tre metodi statici che restituiscono il file di default, i mime configurati e la directory.
Tutte e tre ciclano gli elementi della proprietà desiderata e trovano eventuali corrispondenze.

```
''' <summary>
''' Legge i file di default
''' </summary>
''' <param name="LocalDir">la directory
                                    locle</param>
''' <param name="ErrMessage">messaggi di
                                   errore</param>
''' <returns>il file di default</returns>
Public Shared Function
ReadDefaultFileName(ByVal LocalDir As String, ByRef
              ErrMessage As String) As String
    Dim _DEFAULTFILES As StringCollection =
        CType(ServerSettings.Default.DefaultFiles,
                            StringCollection)
    For Each fileName As String In _DEFAULTFILES
        If (File.Exists(LocalDir + fileName)) Then
            Return fileName
        End If
    Next
    Return ""
End Function

''' <summary>
''' Legge i tipi mime configurati
''' </summary>
''' <param name="RequestedFile">l file</param>
''' <param name="ErrMessage">messaggi di
                                   errore</param>
''' <returns>il tipo mime</returns>
Public Shared Function
ReadConfiguredMimes(ByVal RequestedFile As String,
         ByRef ErrMessage As String) As String
    Dim fileExtension As String = ""
```

```
        Dim mimeExtension As String = ""
        Dim mimeType As String = ""
        Dim _MIMETYPES As StringCollection =
                CType(ServerSettings.Default.MimeTypes,
                                        StringCollection)
        For Each mimeTypeDef As String In
                                              _MIMETYPES
            Dim mimmeArray() As String =
                         mimeTypeDef.ToLower().Split(" ")
            mimeExtension = mimmeArray(0)
            mimeType = mimmeArray(1)
            If (mimeExtension = fileExtension) Then
                Return mimeType
            End If
        Next
        Return ""
    End Function

    ''' <summary>
    ''' Legge il path del sito
    ''' </summary>
    ''' <param name="DirectoryName">nome
                                       directory</param>
    ''' <param name="Messages">messaggi di
                                          output</param>
    ''' <returns>la path</returns>
    Public Shared Function ReadSitePath(ByVal
 DirectoryName As String, ByRef Messages As List(Of
                                  String)) As String
        Dim virtualDirectory As String = ""
        Dim realDirectory As String = ""
        Messages = New List(Of String)()
        Dim _DIRECTORIES As StringCollection =
               CType(ServerSettings.Default.Directories,
                                        StringCollection)

        For Each directoryInfo As String In
                                            _DIRECTORIES
            Dim directoryArray() As String =
                       directoryInfo.ToLower().Split(" ")
            virtualDirectory = directoryArray(0)
            realDirectory = directoryArray(1)
            If (virtualDirectory = DirectoryName) Then
                Messages.Add("INFO\tDirectory virtuale:
                                     " & virtualDirectory)
                Messages.Add("INFO\tDirectory: " &
                                           DirectoryName)
                Messages.Add("INFO\tDirectory fisica: "
                                         & realDirectory)
                Return realDirectory
            End If
        Next
        Return ""
    End Function
```

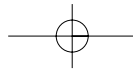
Ad esempio, il metodo **ReadConfiguredMimes** effettua un ciclo sulla *Collection* _MIMETYPES e controlla se tra i tipi mime supportati dal server c'è quello richiesto. Gli altri due metodi funzionano in modo analogo.
Esaminiamo, quindi, il cuore del server: la classe **HTTPServerManager**.

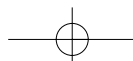
Il costruttore

```
    Public Sub New(ByVal Messages As ListBox, ByVal
                     Ip As String, ByVal Port As Integer)
        _Messages = Messages
        _IP = IPAddress.Parse(Ip)
        _Port = Port
    End Sub
```

Quando viene creata una nuova istanza della classe, vengono valorizzate le proprietà private **_IP** di tipo IPAddress e **_Port** di tipo intero.
La *ListBox* **Messages** viene utilizzata per visualizzare i messaggi del server.
L'avvio del server avviene tramite il metodo **ServerStart**.

```
    Public Sub ServerStart()

        Try
            _Listener = New TcpListener(_IP, _Port)
            _Listener.Start()
            _ListenerThread = New Thread(AddressOf
                                                  Listen)
            _ListenerThread.Start()

            AddMessage("ACTION Web Server Avviato...
                          " & DateTime.Now.ToString())

        Catch e As Exception
            AddMessage("ERROR Errore: " & e.Message)
            _ListenerThread.Abort()
        End Try

    End Sub
```

Viene istanziato un oggetto di tipo TcpListener sulla porta specificata e quindi un Tread che fa partire il metodo Listen.

MIME TYPES

Un tipo MIME (*Multipurpose Internet Mail Extensions*) identifica il tipo di dato che viene inviato attraverso i protocolli HTTP o SMTP. E' uno standard per il formato dei documenti scambiati su Internet. All'inizio era utilizzato per la posta elettronica e si limitava a messaggi in formato ASCII. Quando due programmi si scambiano informazioni attraverso la rete, il mittente deve specificare che tipo di messaggio sta inviando secondo lo standard MIME, così facendo, il programma che lo riceve sa come trattarlo e può decodificarlo in maniera opportuna.

```
Public Sub Listen()

    Dim requestedFile As String = ""
    Dim httpErrorMessage As String = ""
    Dim localDirectory As String = ""
    Dim physicalFilePath As String = ""

    While (True)

        Dim Sock As Socket =
                         _Listener.AcceptSocket()

        If (Sock.Connected) Then

            AddMessage("INFO Client connesso... " &
                         DateTime.Now.ToString())
            AddMessage("INFO Client IP:" &
                         Sock.RemoteEndPoint.ToString())

            'ricevo la richiesta
            Dim buf As String = ""
            Dim methodSupported As Boolean =
                         ReceiveRequest(Sock, buf)
            If (Not methodSupported) Then
                Return
            End If

            'leggo la richiesta
            Dim httpVersion As String = ""
            Dim theRequest As String =
                         ParseRequest(buf, httpVersion)
            'La directory
            Dim directoryExists As Boolean =
        GetDirectory(theRequest, localDirectory, Sock,
                         httpVersion)
            If (Not directoryExists) Then
                Continue While
            End If

            'Leggo il nome del file
            Dim fileExists As Boolean =
      GetFile(theRequest, requestedFile, localDirectory,
                         Sock, httpVersion)
            If (Not fileExists) Then
                Return
            End If

            'Mime Type
            Dim errMessage As String = ""
            Dim mimeType As String =
              HTTPConfigReader.ReadConfiguredMimes
                         (requestedFile, errMessage)
            If (errMessage <> "") Then
                AddMessage(errMessage)
            End If

            'Costruisco il percorso fisico
            physicalFilePath = localDirectory &
```

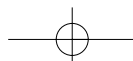
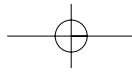
```
                         requestedFile
            AddMessage("INFO File richiesto: " &
                         physicalFilePath)

            If (File.Exists(physicalFilePath)) Then
                SendFileToBrowser(physicalFilePath,
                         Sock, httpVersion, mimeType)
            Else
                httpErrorMessage = "<h3>Errore 404.
                         File non trovato</h3>"
                SendHeader(httpVersion, "",
     httpErrorMessage.Length, " 404 Not Found", Sock)
SendToBrowser(Encoding.ASCII.GetBytes
                         (httpErrorMessage), Sock)
                AddMessage("WARN File non trovato")

            End If
            Sock.Close()
        End If

    End While

End Sub
```

È il vero *entry point* dell'applicazione, mette l'oggetto in attesa di richieste tramite un ciclo infinito.

Come prima cosa, viene creato un **Socket**, ovvero un canale di comunicazione per consentire il flusso dei dati, tramite il metodo **AcceptSocket** dell'oggetto *_Listener*.

```
Dim Sock As Socket = _Listener.AcceptSocket()
```

A questo punto, quando avviene una connessione, il server segue il seguente flusso:

- riceve la richiesta: *ReceiveRequest*
- processa la richiesta: *ParseRequest*
- legge la directory: *GetDirectory*
- legge il nome del file: *GetFile*
- legge il mime type: HTTPConfigReader.Read *ConfiguredMimes*
- se tutto va a buon fine, invia il file al browser: *SendFileToBrowser*

Vediamo alcuni dei metodi in dettaglio:

IPADDRESS

La classe IPAddress del Framework rappresenta un indirizzo IP. Un indirizzo IP identifica univocamente un dispositivo all'interno di una rete TCP/IP. Nella versione IPv4 è formato 32 bit e rappresentato con quattro numeri decimali ciascuno rappresentante un byte da 0 a 255 (ad esempio 127.0.0.1). Nella versione IPv6 è formato da 128 bit ed è rappresentato con 8 numeri esadecimali tra 0 e 65535.

NOTE

IL CODICE ALLEGATO

Gli esempi nell'articolo sono in Visual Basic; per chi fosse interessato, nel CD allegato è presente tutto il codice sorgente anche in C#.

Il metodo **ReceiveRequest**

```
    Private Function ReceiveRequest(ByVal Sock As
                 Socket, ByRef buf As String) As Boolean

        Dim bReceive(1024) As Byte
        Dim i As Integer = Sock.Receive
                         (bReceive, bReceive.Length, 0)
        buf = Encoding.ASCII.GetString(bReceive)
        If (buf.Substring(0, 3) <> "GET") Then
            AddMessage("WARN E' supportato solo il
                                         metodo GET")
            Sock.Close()
            Return False
        End If
        Return True

    End Function
```

Il metodo riceve in ingresso il *Socket* e restituisce in output una stringa contenente la richiesta per la lettura della quale viene utilizzato il metodo **Receive** della classe *Socket* .

Il metodo **SendFileToBrowser**

```
    Private Sub SendFileToBrowser
        (ByVal physicalFilePath As String, ByVal Sock As
Socket, ByVal httpVersion As String, ByVal MimeType
                                            As String)

        Dim totalBytes As Integer = 0
        Dim fs As FileStream = New
            FileStream(physicalFilePath, FileMode.Open,
                     FileAccess.Read, FileShare.Read)
        Dim reader As BinaryReader = New
                                   BinaryReader(fs)
        Try
            Dim bytes(fs.Length) As Byte
            Dim read As Integer
            'leggo il contenuto del file da inviare al
                                             browser
            While (True)
                read = reader.Read(bytes, 0,
                                       bytes.Length)
                totalBytes = totalBytes + read
                If (read = 0) Then
                    Exit While
                End If
            End While
            'invio l'header http
            SendHeader(httpVersion,
                 MimeType, totalBytes, " 200 OK", Sock)
            'invio il contenuto
            SendToBrowser(bytes, Sock)
        Finally
            reader.Close()
            fs.Close()
        End Try

    End Sub
```

TCPCLIENT

La classe TCPClient fornisce dei metodi per creare un client TCP. Per funzionare ha bisogno di avere un server TCP in ascolto per collegarsi al quale può utilizzare il metodo Connect. Fornisce inoltre il metodo GetStream per ottenere un NetworkStream da cui poter leggere i dati ricevuti dal server. Ulteriori informazioni sono reperibili all'indirizzo http://msdn2.microsoft.com/it-it/library/system.net.sockets.tcpclient(vs.80).aspx

Legge il file dal percorso fisico tramite un **BinaryReader** e lo invia al Browser. Per farlo, utilizza due metodi:

- **SendHeader** che invia l'header HTTP.
- **SendToBrowser** che invia il contenuto al browser.

Ecco il metodo **SendToBrowser**, che riceve in ingresso un array di Byte che provvederà ad inviare tramite il metodo **Send** della classe Socket.

```
    Public Sub SendToBrowser(ByVal SendData As
                     Byte(), ByVal Sock As Socket)
        Dim bytesNum As Integer = 0
        Try
            If (Sock.Connected) Then
                bytesNum = Sock.Send(SendData,
                              SendData.Length, 0)
                If (bytesNum = -1) Then
                    AddMessage("ERROR Errore Socket,
                       impossibile inviare i pacchetti")
                Else
                    AddMessage("INFO bytes inviati: " &
                                          bytesNum)
                End If
            Else
                AddMessage("ERROR Connessione
                                        interrotta")
            End If
        Catch e As Exception
            AddMessage("ERROR Errore: " & e.Message)
        End Try
    End Sub
```

I metodi non esposti nell'articolo sono comunque presenti nel CD allegato. Non ci resta che compilare il progetto e far partire il nostro HTTP Server! Abbiamo intravisto le potenzialità di comunicazione offerte dai **Socket**, con cui si può realizzare praticamente qualsiasi protocollo di comunicazione. Il Framework .Net fornisce diverse classi per eseguire compiti specifici, come l'invio di richieste http o le connessioni a dei server ftp; vediamo un po' come fare.

HTTP

Il protocollo HTTP (*Hyper Text Transfer Protocol*) è un protocollo di livello applicativo utilizzato per la trasmissione delle informazioni su web. La comunicazione HTTP viene realizzata tramite

014-020 29-06-2007 17:27 Pagina 19

delle sequenze richiesta/risposta tra client e server. A differenza degli altri protocolli dello stesso livello, HTTP lavora in modalità disconnessa: questo vuol dire che non si istaura un canale di comunicazione permanente tra client e server ma, ad ogni richiesta, viene aperto e poi richiuso il canale di comunicazione.
Il Framework .Net fornisce due classi base per la gestione delle comunicazioni tramite HTTP: *HttpWebRequest* and *HttpWebResponse*.
Una particolarità di queste due classi è che non vengono istanziate direttamente ma utilizzano un *design pattern* creazionale chiamato **Factory**. Prendiamo come esempio la classe *HttpWebRequest*. Per istanziare un oggetto di questo tipo, dobbiamo passare attraverso la sua classe "*Creator*", ovvero la **WebRequest**, la quale, tramite il metodo **Create**, può restituire un oggetto di tipo *HttpWebRequest*
Implementiamo un downloader di files che prende una lista di indirizzi web e, tramite delle richieste HTTP, ottiene in risposta degli *Stream* che utilizza per copiare i file in una cartella locale.

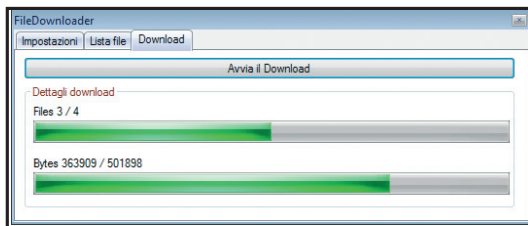



Fig. 2: La MainForm del Downloader

Tralasciamo la costruzione della MainForm - presente nel CD allegato - ed analizziamo il cuore del programma rappresentato da due classi:

- **HttpRequestManager.vb**: che contiene i metodi per effettuare le richieste HTTP al server e ricevere uno *Stream* in risposta.
- **HttpDownloadManager.vb**: che contiene i metodi per salvare lo *Stream* su file.

La classe **HttpRequestManager** è la seguente:

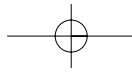
```
Imports System.Net
Imports System.IO

Public Class HttpRequestManager

    ''' <summary>
    ''' la risposta Http
    ''' </summary>
    Private _Response As HttpWebResponse
    ''' <summary>
    ''' la richiestaHttp
    ''' </summary>
    Private _Request As HttpWebRequest

    Private _FileSize As Long
    ''' <summary>
    ''' la dimensione del file
    ''' </summary>
    Public ReadOnly Property FileSize()
        Get
            Return _FileSize
        End Get
    End Property

    ''' <summary>
    ''' Costruttore
    ''' </summary>
    ''' <param name="url">l'url del file da
                                        scaricare</param>
    Public Sub New(ByVal url As String)
        _Request = CType(WebRequest.Create(url),
                                        HttpWebRequest)
        _Response = CType(_Request.GetResponse(),
                                        HttpWebResponse)
        _FileSize = _Response.ContentLength
    End Sub
    ''' <summary>
    ''' Chiude lo stream
    ''' </summary>
    Public Sub Close()
        _Response.Close()
    End Sub
    ''' <summary>
    ''' Chiude lo stream
    ''' </summary>
    Public Function GetDownloadStream() As Stream
        Return _Response.GetResponseStream()
    End Function

End Class
```

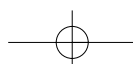
Contiene un costruttore per generare la richiesta ed i metodi per leggere lo *Stream* di risposta.
Il costruttore della classe invia una richiesta HTTP tramite il metodo **Create** della classe **WebRequest** che restituisce un oggetto "castabile" in *HttpWebRequest*.

```
_Request = CType(WebRequest.Create(url),
                                HttpWebRequest)
```

Una volta create la richiesta, viene utilizzato il metodo **GetResponse** della classe **HttpWebRequest** per ottenere un oggetto che "castiamo" in *HttpWebResponse*.

```
_Response = CType(_Request.GetResponse(),
                                HttpWebResponse)
```

Importiamo i namespace:

STREAM

Uno Stream rappresenta un flusso di dati in Input/output come una sequenza di Byte. La classe Stream del Framework .Net è una classe astratta alla base di tutti i tipi di Stream. Questi possono essere ad esempio FileStream (per la lettura e scrittura di file) o NetworkStream (per l'invio e la ricezione di dati mediante socket).

- **System.Net**: contiene le classi per effettuare le richieste HTTP.

- **System.IO**: per scrivere su file system.

Tutto parte dal metodo statico (*Shared* in VB) *Download* della classe **HttpDownloadManager** che come prima cosa istanzia un oggetto di tipo *HttpRequestManager* che, come visto sopra, invia la richiesta http e riceve una risposta.

Vediamolo in dettaglio:

```
''' <summary>
''' Download dei file
''' con impostazione dei valori in una text
''' e avanzamento progress
''' </summary>
''' <param name="FileUrl">l'url del file da
                                        scaricare</param>
''' <param name="SaveDestination">la destinazione
                                su cui salvare il file</param>
''' <param name="prg_bytes">la progress bar da far
                                         avanzare</param>
''' <param name="txt_bytes">la label da
                                       aggiornare</param>
Public Shared Sub Download(ByVal FileUrl As String,
  ByVal SaveDestination As String, ByVal prg_bytes As
                   ProgressBar, ByVal txt_bytes As Label)

    Dim downloader As HttpRequestManager = New
                             HttpRequestManager(FileUrl)
    Dim destination As FileStream =
              File.Open(SaveDestination, FileMode.Append,
                                       FileAccess.Write)
    Try
        Dim buffer(1024) As Byte
        Dim bytesRead As Integer = 0
        Dim totalDownloaded As Long = 0
        Dim totalFileSize As Long = downloader.FileSize

        If (Not prg_bytes Is Nothing) Then
            prg_bytes.Minimum = 0
            prg_bytes.Maximum =
                   Convert.ToInt32(totalFileSize / 1024)
            prg_bytes.Step = 1
            prg_bytes.Value = 0
        End If
        'leggo il contenuto della risposta e lo scrivo
                             nello stream di destinazione
        While (True)
            If (Not txt_bytes Is Nothing) Then
                txt_bytes.Text = String.Format("Bytes
            {0} / {1}", totalDownloaded, totalFileSize)
            End If
            'Ottengo lo stream di risposta
            Dim downloadStream As Stream =
                          downloader.GetDownloadStream()
            bytesRead = downloadStream.Read
                                     (buffer, 0, 1024)
            If (bytesRead > 0) Then
                totalDownloaded += bytesRead
                'scrivo sulla destinazione
                destination.Write(buffer, 0, bytesRead)
                If (Not prg_bytes Is Nothing) Then
                    prg_bytes.PerformStep()
                End If
            Else
                Return
            End If
        End While
    Finally
        downloader.Close()
        destination.Close()
    End Try
End Sub
```

Una volta ricevuta la risposta, occorre effettuare il download del file. A tale scopo, utilizziamo un *FileStream* (*destination*).
Viene, quindi, letto lo *Stream* ottenuto tramite il metodo **GetResponseStream** della classe *Http-WebResponse* e salvato sul file di destinazione creato precedentemente.
Il programma principale non fa altro che leggere una lista di indirizzi HTTP ed utilizzare il metodo Download per ciascuno di essi.

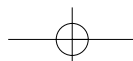
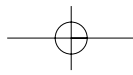
***Fig. 4:* Schema di funzionamento del downloader**

CONCLUSIONI

Come abbiamo potuto notare nel corso dell'articolo, il Framework fornisce una notevole quantità di classi per la gestione dei protocolli di rete e la realizzazione di applicazioni che li utilizzano. Alcune di esse sono molto semplici, altre richiedono una maggiore conoscenza delle tecniche di programmazione. Sono, comunque, classi molto flessibili e, con un po' di fantasia da parte nostra, riescono a farci gestire qualsiasi aspetto della programmazione di rete.

Carmelo Scuderi

L'AUTORE

Carmelo Scuderi è ingegnere informatico. Si occupa di sviluppo software in ambiente .Net per una società di informatica di Milano. Gestisce un sito ricco di script e manuali per chi si affaccia al mondo della programmazione web (www.morpheusweb.it).

AJAX VISTO CON L'OCCHIO DEL SERVER

NORMALMENTE SIAMO ABITUATI A PENSARE AD UN MECCANISMO IN CUI UN SERVER RISPONDE ALLE DOMANDE DI UN CLIENT. E SE INVERTISSIMO I RUOLI? ECCO COME FARE. INVIAMO I DATI AL CLIENT IN MANIERA AUTONOMA QUANDO SONO DISPONIBILI...

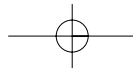
Nonostante l'apparenza, Lightstreamer è un prodotto italiano al 100%, e come sicuramente emergerà nel corso di questa trattazione, esso rappresenta una novità di assoluto rilievo non soltanto in ambito nazionale, ma più in generale nell'intero mondo dello sviluppo web *Ajax-based*.
Negli scorsi numeri si è parlato di Dojo come uno dei pochissimi framework in grado di implementare il paradigma *AJAX-Comet*, basato sul *pushing* lato-server: un approccio, questo, che sovverte il classico funzionamento delle applicazioni Ajax tradizionali, consentendo invece un dialogo tra client e server molto più vantaggioso e performante.
Ebbene, la libreria Lightstreamer sfrutta in pratica gli stessi principi, i cui indubbi vantaggi, occorre ricordarlo, possono essere qui di seguito sintetizzati:

- innanzitutto, non abbiamo più un continuo bombardamento (polling) di richieste nei confronti del server: ora piuttosto quest'ultimo diventa "collaborativo", e quindi è in grado da solo di prendere delle iniziative al verificarsi di determinati eventi, con grande risparmio in termini di traffico di banda e risorse del sistema;

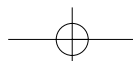
- la soluzione Comet, peraltro, al pari di Ajax, non necessita sul client dell'installazione di alcun componente particolare (né ActiveX, né applet, né plugin vari, né Flash, che pure sono individualmente supportati dal framework in altri contesti, dei quali faccio alcuni accenni in un box apposito), mentre richiede unicamente la presenza di un comune browser per il web.

Avremo presto modo di studiare più nel dettaglio il particolare approccio ad Ajax adottato da Lightstreamer, che rappresenta a tutt'oggi lo stato dell'arte della programmazione asincrona.

Prima, però, è il caso di soffermarci un attimo sulle reali motivazioni che possono spingere un programmatore a preferire una soluzione complessa ed avanzata come Comet, rispetto invece al suo "rivale" più famoso Ajax.
È bene puntualizzare, a questo riguardo, che per la maggior parte delle *web-application*, già funzionanti con successo secondo lo stile Ajax, non vi è alcun vantaggio significativo in una "rivisitazione" in chiave Comet.
Quest'ultimo approccio, infatti, si rende necessario soltanto in applicazioni che richiedono un'interazione con il server in *real-time*: mi riferisco, evidentemente, al commercio *on-line* di valori finanziari, le cui quotazioni devono essere aggiornate in tempo reale, oppure ai siti che visualizzano risultati sportivi, ecc...
Va rimarcato, inoltre, che i pregi di Lightstreamer non sono solo legati al suo approccio decisamente "futuristico" nei confronti di Ajax, ma si spingono ben oltre, investendo anche problematiche legate alla sicurezza e alla scalabilità.
Mediante opportune impostazioni, infatti, è possibile assegnare un tetto massimo di occupazione di banda per ogni canale di *streaming*, applicando al flusso di dati dei particolari filtri dinamici basati su sofisticati algoritmi euristici. Questo serve a gestire in modo oculato il traffico di banda per ogni utente, che altrimenti potrebbe subìre aumenti incontrollati in applicazioni, come quelle basate sul *pushing* in *real-time*, particolarmente esposte a questo rischio. Allo stesso modo è anche possibile intervenire sulla massima frequenza di *update* delle varie entità oggetto di monitoraggio da parte degli utenti. Il Lightstreamer server, inoltre, è in grado di adattarsi, in modo intelligente, alle condizioni generali e allo stato di funzionamento della rete, adeguando la portata del flusso informativo alla quantità massima di dati che il sistema è capace di gestire in ogni dato momento,

Conoscenze richieste
Javascript, Ajax, Dojo framework, elementi di Java per web-application

Software
Un browser web, Lightstreamer framework, Lightstreamer Dojo Demo, Dojo framework (versione Ajax), J2 SDK, un webserver

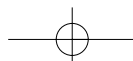
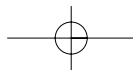
Impegno

Tempo di realizzazione

al fine di prevenire inconvenienti facilmente immaginabili.
Un altro grande punto di forza è poi rappresentato dalla particolare architettura su cui è basato il server, che gestisce le varie connessioni mediante un meccanismo asincrono e dinamico, non più basato sulla rigida relazione "uno-a-uno" tra ogni connessione e il suo corrispondente *thread*, come invece avviene in generale nei *webserver* tradizionali. In questo modo si possono gestire molte più connessioni a fronte dei *thread* effettivamente impiegati (ricordiamo che a differenza di Ajax, in questo caso, ogni connessione rimane in vita per lungo tempo, quindi non sarebbe possibile adottare il sistema tradizionale).
Infine, occorre precisare che Lightstreamer consente anche di aumentare la scalabilità adottando la soluzione dei server in *cluster*, mediante l'applicazione di un apposito sistema di bilanciamento del carico.

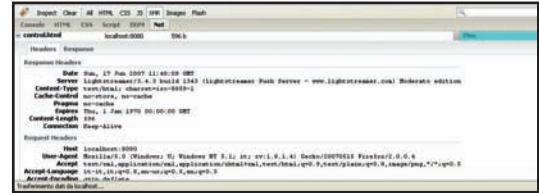
Paradigma	Metodo di invio dati rispetto all'utente	Metodo di invio dati rispetto al browser
Web-application tradizionale - refresh delle pagine	sincrono	sincrono
Classic Ajax Application - polling periodico	asincrono	sincrono
Smart Ajax Application: - es. : polling asincrono (è adottato da Lightstreamer in alcune particolari situazioni)	asincrono	parzialmente asincrono
Streaming Ajax Application: - es. streaming Ajax (è quello normalmente adottato da Lightstreamer)	asincrono	asincrono

STREAMING AJAX: IL "TRUE-PUSH/ STREAMING PARADIGM"

Per capire bene quali sono i reali vantaggi che caratterizzano il particolare approccio Comet-based di Lightstreamer, occorre fare un passo indietro e analizzare prima le diverse tipologie di interazione tra client e server attualmente esistenti.
Ecco quindi uno schema riepilogativo nella **tabella numero 1.**
Il tipo di interazione può essere osservato naturalmente sotto due differenti punti di vista: quello dell'utente e quello del browser. Ad esempio, se l'utente, per aggiornare i dati visualizzati sullo schermo, deve effettuare manualmente una qualche azione, oppure se durante la fase di aggiornamento l'interfaccia grafica si presenta bloccata, allora parleremo di "interazione sincrona rispetto alle azioni dell'utente".
Se invece prendiamo come riferimento il browser, e constatiamo che esso, per controllare la presenza di aggiornamenti, deve tutte le volte inviare una nuova richiesta al server ed attendere una risposta da parte di quest'ultimo, anche senza l'intervento manuale dell'utente, allora possiamo parlare di "interazione sincrona rispetto alle azioni del browser".
Ne consegue che l'unico vantaggio apportato dal "**polling periodico**" basato su Classic Ajax, rispetto al Refresh tradizionale delle pagine, è proprio quello di rendere asincrona l'interazione con l'utente (ad esempio, dopo aver inviato la prima richiesta, l'utente può continuare a interagire tranquillamente con l'interfaccia grafica, anche se il browser sta lavorando in "background"). Il metodo di scambio dati tra browser e server, tuttavìa, continua ad essere sincrono, oltre a comportare il possibile invìo di una molteplicità di richieste totalmente inutili, e tutto ciò, ovviamente, si riflette in un rallentamento generale delle prestazioni dell'applicazione e in un aumento spropositato del carico di sistema.
Una soluzione, seppur solo parziale, agli appena citati inconvenienti, ci viene proposta dal cosiddetto "**polling asincrono**": in questo caso, non abbiamo più una serie di richieste inviate al server a intervalli predefiniti, ma ne viene inviata una soltanto. Il server, inoltre, non ritorna subito la risposta ricevuta, ma resta in attesa che si verifichi il particolare

022-027 29-06-2007 16:48 Pagina 24

evento richiesto dal client (ad esempio, nel caso di una chat, la presenza di nuovi messaggi da notificare ai vari utenti). Solo allora, quindi, restituirà al client la risposta, evitando quindi la generazione di traffico inutile. Una volta ricevuta la risposta, a questo punto, il client invierà una nuova richiesta al server, e si rimetterà in funzione il meccanismo appena descritto.
Tuttavìa, anche in questo caso, è evidente che per ottenere qualsiasi aggiornamento della pagina occorre sempre un ciclo completo di "domanda/risposta" (**polling cycle**), poco vantaggioso nel caso di applicazioni basate su *update* molto frequenti. Ed è per questo motivo che un simile approccio viene considerato asincrono solo parzialmente (dal punto di vista del browser), mentre il metodo adottato da Lightstreamer, che sfrutta una sola connessione tenuta in vita per tutta la durata dell'applicazione, rappresenta una valida soluzione anche agli inconvenienti appena esposti.
In particolare, i pregi dello "**streaming Ajax**" (detto anche **Comet – forever frame**), rispetto al "**polling asincrono**" possono essere così sintetizzati:

- la latenza di rete è praticamente ininfluente, non essendo necessario un nuovo "**polling cycle**" per ogni aggiornamento, quindi è possibile spingersi ai massimi valori di frequenza di *update*;

- anche il traffico di banda è molto ridotto, visto che le pesanti intestazioni HTTP dei "**polling cycles**" sono abolite, e che le risposte sono inviate ai client solo quando ve ne è realmente la necessità;

- tutto ciò, naturalmente, ha una ricaduta positiva sull'intero sistema, che si concretizza in un incremento delle prestazioni dell'applicazione e in un notevole alleggerimento di carico.

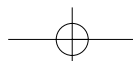
Tramite lo "**streaming Ajax**" possiamo facilmente sfruttare al massimo la banda disponibile, generando persino sofisticati grafici dinamici basati totalmente su dati prelevati in *real-time*. Mi riferisco, evidentemente, alla **Lightstreamer Dojo Demo**, un'applicazione scaricabile separatamente rispetto al framework, che vedremo in seguito come installare e configurare opportunamente sul nostro pc casalingo.
È importante, altresì, sottolineare che un simile approccio alle comunicazioni asincrone è totalmente inutile in alcune situazioni particolari, dove ad esempio sono presenti dei componenti (proxy server, software antivirus) che bloccano temporaneamente i contenuti inviati ai client per ispezionarli. Ebbene, dal momento che in questi casi lo *streaming* non è proprio realizzabile (neppure se attuato nei confronti di dispositivi desktop o di applet Java, beninteso), Lightstreamer prevede una funzionalità (la **Stream-Sense**) che è in grado di accorgersi autonomamente della presenza di tali componenti, e di adottare, nei casi in questione, un approccio ad Ajax più consono ("**smart polling Ajax**"). Anche in questo caso, comunque, va da sé che le altre importanti prerogative di Lightstreamer (controllo di banda, applicazioni di filtri e gestione delle frequenze di *update*), rimangono pienamente preservate.

CHE COSA È FIREBUG?

Firebug, nota estensione di Firefox utile per gli sviluppatori di web-application, consente di rappresentare con grande efficacia la sequenza delle chiamate Ajax generate da un browser. L'immagine sopra riportata, in particolare, si riferisce ad un'applicazione che implementa il paradigma "Comet – Long Poll", basato sul "polling asincrono": facilmente si possono vedere le long-polled connections, che restano sul server per lungo tempo in attesa di ricevere la risposta desiderata. Non appena una di queste richieste riceve una risposta, questa viene subito rispedita al client, che invia immediatamente una nuova richiesta. L'icona in basso a sinistra, che nella realtà è un'immaginetta animata, rappresenta con grande efficacia una richiesta inviata al server e temporaneamente ferma, in attesa di ricevere una risposta (o che termini la vita massima della connessione).
Lightstreamer tiene aperta per ogni client una sola connessione, in modo permanente, inoltre le risposte sono inviate ai vari client soltanto quando vi sono effettivamente "dati freschi" da comunicare: questo evita la generazione di traffico inutile, rendendo possibile lo streaming dei dati e riducendo notevolmente il carico di sistema.

DALLA TEORIA ALLA PRATICA: INSTALLARE IL FRAMEWORK

Per avvicinarsi al framework, conviene sicuramente prima di tutto provare le tante demo

022-027 29-06-2007 16:48 Pagina 25

on-line, i cui link sono reperibili direttamente sul sito web di Lightstreamer (http://www.lightstreamer.com).
Sul CD allegato è presente, inoltre, la versione Moderato del framework, liberamente scaricabile anche presso il sito (http://www.lightstreamer.com/downloadfree.htm) e installabile in locale sul proprio PC.
E' bene precisare, a questo punto, che l'architettura classica di una web-application basata su Lightstreamer è costituita dai seguenti componenti:

- un *webserver* (per i miei esempi utilizzerò Apache, che è liberamente scaricabile al sito http://www.apache.org), il quale serve unicamente la parte statica dell'applicazione;
- il server Lightstreamer, il quale, invece, si occupa dell'elaborazione lato server e gestisce quindi tutta la parte dinamica.

L'installazione del framework è molto semplice (cito solo i passaggi per l'installazione in Windows): basta scompattare la cartella zippata del framework nel *webserver*, e quindi modificare il file LS.bat presente nella cartella **<cartella di installazione del framework>bin/windows**, in base alla configurazione del proprio sistema operativo.

es.
LS_HOME -> C:\xampp\htdocs\programmi\Lightstreamer (è il percorso della cartella principale del framework, secondo la mia configurazione)

Da questo momento con LS_HOME intenderemo, tra l'altro, la directory principale di Lightstreamer

JAVA_HOME -> C:\Programmi\Java\jdk1.6.0 (è il percorso della cartella principale dell'SDK di Java, che dovrebbe essere presente anche tra le variabili d'ambiente del sistema operativo)

A questo punto, lanciamo il file **Start_LS_as_Application.bat**, e dovrebbe aprirsi una shell simile alla seguente:

La shell appena aperta ci informa che il server di Lightstreamer è attivo e in ascolto sulla porta 8080: puntando, a questo punto, un browser su http://localhost:8080, dovremmo accedere alla videata principale del framework.
Osserviamo, ora, finalmente, il funzionamento della Stock List Demo, la prima applicazione di esempio che troviamo linkata nella suddetta pagina.
La cartella principale dell'applicazione (StockListDemo) si trova nella directory **pages**, posta direttamente nella LS_HOME, e contiene tutta logica di elaborazione lato client (in particolare, la cartella LS con la logica Javascript di Lightstreamer).
Il file che si apre puntando il browser sul link alla Stock List Demo è, naturalmente, index.html:

Analizzando il codice della pagina, osserviamo innanzitutto l'inclusione dei file JS principali di Lightstreamer:

```
<script src="ls/lscommons.js"></script>
<script src="ls/lspushpage.js"></script>
```

Ed ecco in seguito le righe che avviano e configurano il motore di pushing:

```
///////////////////////////////PushPage
                                        Configuration

        var lsPage = new PushPage();
        //
 lsPage.context.setDebugAlertsOnClientError(false)
                                                  ;
        lsPage.context.setDomain(null);
        lsPage.onEngineCreation = startEngine;
        lsPage.bind();
        lsPage.createEngine("SLEngine", "ls/");
        //starting subscribing first 15 stocks
        changePage(1);

function startEngine(eng) {
        //
  eng.context.setDebugAlertsOnClientError(false);
        eng.policy.setMaxBandwidth(30);
        eng.policy.setIdleTimeout(30000);
        eng.policy.setPollingInterval(1000);
        eng.connection.setLSHost(null);
        eng.connection.setLSPort(null);
eng.connection.setAdapterName("STOCKLISTDEM
                                          O");
        eng.changeStatus("STREAMING");
}
```

È interessante, in particolare, la riga che imposta il nome del **Data Adapter** ("STOCKLISTDEMO"), ossìa il componente che fornisce i dati all'applicazione. Nel caso specifico, in particolare, trattandosi di un'applicazione di esempio funzionante localmente, il Data Adapter (i cui sorgenti scritti in Java sono disponibili nella directory LS_HOME/DOCS-SDKs\sdk_adapter_java\examples\StockListDemo _DataAdapter\src_adapter) genera i dati casualmente, pur rispettando la coerenza degli stessi e i vincoli di base a cui sono assoggettate le informazioni appartenenti alla negoziazione dei titoli. Va da sé, comunque, che in uno scenario reale, il Data Adapter non genera dati casuali ma si collega direttamente ad un *information provider* di mercato.

La funzione sotto riportata, infine, consente di selezionare e visualizzare le diverse pagine (di 15 elementi cadauna), in cui sono ripartite le varie "entità" oggetto di monitoraggio: ciò avviene istanziando un oggetto **OverwriteTable**, il quale è legato ad una vera e propria tabella fisica che riceve le informazioni in real-time e le inserisce nelle celle associate, sovrascrivendo di volta in volta i vecchi valori:

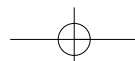
```
function changePage(groupNumber) {
        if (groupNumber == 1) {
                groupArray = page1;
document.getElementById("switchP1").style.displa
                                     y = "none";
document.getElementById("switchP2").style.displa
                                         y = "";
        } else if (groupNumber == 2) {
                groupArray = page2;
document.getElementById("switchP1").style.displa
                                         y = "";

document.getElementById("switchP2").style.displa
                                     y = "none";
        }

        var groupString = "";
        var st = 1;
        for (st = 1; st <= 15; st++) {
                groupString += "item" +
                          groupArray[st-1] + " ";
        }
        actGroup = groupArray;
        var newTable = new
   OverwriteTable(groupString, schema, "MERGE");
        newTable.setSnapshotRequired(true);
newTable.setRequestedMaxFrequency(0.5);
        newTable.setClearOnDisconnected(true);
                                //default is false
        newTable.setClearOnRemove(true);
                                //default is false
        newTable.onItemUpdate = updateItem;
        newTable.onChangingValues =
                                    formatValues;
        newTable.setPushedHtmlEnabled(true);
        lsPage.addTable(newTable, "1");
}
```

L'oggetto OverwriteTable, come si può chiaramente vedere, esegue degli event handler su aggiornamento delle varie entità ("updateItem"), e in altre situazioni, che non riporto per ragioni di spazio, ma che si possono facilmente trovare direttamente sul file index.html della demo.

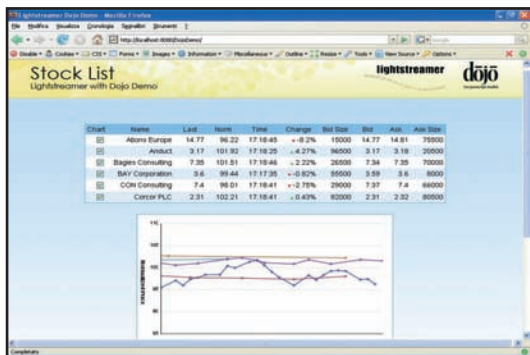
A questo punto, avendo analizzato la Stock List Demo, possiamo installare un'altra applicazione particolarmente interessante, che ci dà un'immediata visione delle straordinarie potenzialità del framework: la Lightstreamer Dojo Demo.

QUALI VERSIONI DEL FRAMEWORK SONO DISPONIBILI ?

Attualmente sono disponibili 4 differenti versioni del framework: Moderato, Allegro, Presto e Vivace. L'unica versione Free, in questo momento, è la Moderato, che manca di di alcune funzionalità, come il controllo di banda o il clustering, ma consente ugualmente di testare le principali applicazioni e prendere visione della complessità del framework. Lightstreamer Moderato funziona automaticamente in "unregistered mode", finché l'utente non richiede una regolare licenza Free per il proprio MAC Address direttamente sul sito: in assenza di tale licenza, compariranno periodicamente dei messaggi (nella shell e nel browser) che invitano l'utente a effettuare la registrazione.

022-027 29-06-2007 16:48 Pagina 27

INSTALLIAMO LA LIGHTSTREAMER DOJO DEMO

Liberamente scaricabile all'indirizzo http://www.lightstreamer.com/share/LS_Dojo.zip, e presente peraltro anche nel CD allegato alla rivista, la Lightstreamer Dojo Demo è la chiara dimostrazione delle straordinarie potenzialità dello "**streaming Ajax**": tramite essa possiamo generare facilmente un grafico in *real-time* contenente diverse serie di dati, utilizzando soltanto un comune browser con Javascript abilitato e niente di più.
Uno studio approfondito dell'applicazione richiede, tuttavìa, la conoscenza del motore di generazione grafici di Dojo (http://dojotoolkit.org), che non può naturalmente essere affrontata in questa sede: mi limiterò, pertanto, a fornire indicazioni per l'installazione in locale dell'applicazione, e a dare anche qualche elemento utile per uno studio autonomo delle varie funzionalità trattate.
Le modalità di installazione della **LightStreamer Dojo Demo** sono le seguenti:

- dopo aver scaricato il pacchetto, innanzitutto, occorre scompattarlo all'interno della LS_HOME/pages;
- poi bisogna copiare i file contenuti in LS_Home/DOCS-SDKs/sdk_client_web/lib nella LS_Home/pages/DojoDemo/LS (se quest'ultima non esiste occorre crearla);
- infine, occorre inserire in LS_Home/pages/DojoDemo una distribuzione Dojo in versione Ajax (in un box laterale fornisco indicazioni su come reperire tale componente), avendo cura di rinominare quest'ultima semplicemente in "Dojo" (oppure cambiare i riferimenti nel file index.html);
- a questo punto, possiamo avviare il Lightstreamer server e puntare il browser su http://localhost:8080/DojoDemo, per vedere in funzione la nostra demo.

La Dojo Demo, in particolare, condivide con la Stock List Demo il Data Adapter, quindi il motore di generazione dinamica dei dati. Non si basa invece sulla OverwriteTable, preferendo piuttosto optare per la NonVisualTable, un componente costituito sempre da una tabella, per l'organizzazione logica dei dati, ma non legato ad una struttura HTML reale.
A chi volesse approfondire autonomamente lo studio della Lightstreamer Dojo Demo, consiglio di visitare direttamente il seguente sito web, che fornisce una descrizione completa ed esaustiva del funzionamento dell'applicazione: http://app.lightstreamer.com/Dojo-Demo.

E' POSSIBILE INTEGRARE LIGHTSTREAMER CON ALTRE TECNOLOGIE ?

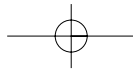
Alla pagina http://www.lightstreamer.com/demos.htm, è possibile avere un assaggio della grande quantità di tecnologie con cui Lightstreamer può facilmente integrarsi: .Net, Java, Excel, Flash, Java Midlet (per cellulari e palmari compatibili con Java), applicazioni Html e Javascript ottimizzate per piccoli schermi (visualizzabili, queste ultime, tramite il browser Opera versione 8 o superiore).

CONCLUSIONI

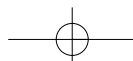
Abbiamo visto che Lightstreamer rappresenta la soluzione più evoluta per tutte quelle *web-application* che necessitano di ricevere aggiornamenti in real-time. Aggiungiamo, a questo punto, anche se fuori tema rispetto allo scopo dell'articolo, che tale tecnologia non è solo specifica per il web, ma si può perfettamente adattare anche alle applicazioni tradizionali (es. desktop), che verrebbero quindi a beneficiare di tutte le avanzate funzionalità legate alla sicurezza e alla scalabilità illustrate inizialmente.
Insomma, è evidente che Lightstreamer è un vero e proprio concentrato di innovazioni, che ha tutte le carte in regola per segnare una nuova tappa nel processo di evoluzione dell'I.T. .

Enrico Viale

L'AUTORE

Enrico Viale è specializzato nello sviluppo di applicazioni sìa web-oriented che desktop. Chi desidera contattarlo per chiarimenti riguardo all'articolo, o per qualsiasi altro motivo, può farlo all'indirizzo enrico.viale@yahoo.it

COME FACCIO A INSTALLARE DOJO FRAMEWORK ?

Per installare correttamente la Lightstreamer Dojo Demo, una delle applicazioni di esempio proposte in questo articolo, bisogna disporre preventivamente del Dojo framework, il quale è scaricabile al sito http://dojotoolkit.org. L'installazione di questo componente è molto semplice: basta scompattare la cartella zippata nella posizione del webserver indicata nel testo. Tenere presente, tuttavìa, che la distribuzione da installare deve essere in versione Ajax: es. dojo-0.4.2-ajax

030-037 29-06-2007 16:05 Pagina 30

UTILIZZIAMO AJAX SENZA XML

XML PUÒ ESSERE UNA SOLUZIONE COMODA PER MOLTI PROBLEMI, MA IN QUALCHE CASO POTREBBE ESSERE UTILE LAVORARE CON ALTRI TIPI DI FORMATI, AD ESEMPIO JSON O TESTO. VEDIAMO COME SVINCOLARE AJAX DALLA SCHIAVITÙ DI XML

Sono trascorsi più di due anni da quando Jesse James Garrett coniò il termine Ajax. Questa fantastica tecnologia, in realtà, consiste in un insieme di tecniche atte a permettere una comunicazione asincrona tra client e server ed il successivo aggiornamento dinamico degli elementi di una pagina web. Da allora sono stati pubblicati diversi articoli che avevano come target Ajax. Quest'articolo ha lo scopo di mettere assieme i vari tipi di response possibili ed illustrare come gestirli uno ad uno. In particolare, vedremo come manipolare i tre tipi di response più utilizzati in ambito Ajax, ossia:

- HTML
- XML
- JSON

Oltre a questi, illustreremo come gestire response di tipo Plain Text con un protocollo da noi ideato. Questo a riprova del fatto che Ajax è indipendente da XML a dispetto del suo significato originale (AJAX = Asynchronous JavaScript + XML).

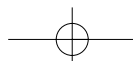
CREAZIONE DELL'OGGETTO XMLHTTPREQUEST

Prima di analizzare i tipi di response citati, ci occorre creare una funzione atta alla costruzione dell'oggetto cuore di Ajax, ossia *XMLHttpRequest*. È tramite quest'oggetto, infatti, che eseguiamo chiamate asincrone al server e riceviamo il successivo response, qualsiasi sia il tipo (Plain Text, HTML, XML o JSON). La funzione *createHttpRequest* sarà utilizzata per tale scopo nel prosieguo dell'articolo. Ecco il codice:

```
// Crea un oggetto di tipo XMLHttpRequest
function createHttpRequest()
```

```
{
    if (typeof XMLHttpRequest != "undefined")
    {  // E' disponibile XMLHttpRequest in modo nativo
        return new XMLHttpRequest();
    }
    else if (window.ActiveXObject)
    {       // E' una versione di IE precedente la 7
        var versions = [
            "MSXML2.XMLHttp.5.0",
"MSXML2.XMLHttp.4.0","MSXML2.XMLHttp.3.0",
"MSXML2.XMLHttp","Microsoft.XMLHttp"
        ];

        // Cerca di istanziare la versione più recente.
        // Se non è disponibile prova via via con quelle più datate
        for (var i = 0; i < versions.length; i++)
        {
            try
            {
                var ret = new ActiveXObject(versions[i]);
                return ret;
            }
            catch (oException)
            {
                // Non fa niente. Va avanti tentando con le altre versioni
            }
        }
    }
    // Se arriva qui allora l'oggetto XMLHttpRequest non è disponibile per il browser in uso
    alert("Il browser in uso è obsoleto. Aggiornarlo con uno più recente");
}
```

La necessità di utilizzare questa funzione sta nel fatto che Firefox ed Opera hanno un modo diverso, rispetto ad Internet Explorer, di istan-

030-037 29-06-2007 16:05 Pagina 31

ziare l'oggetto utilizzato per inviare chiamate Ajax. La funzione controlla se l'oggetto *XMLHttpRequest* è direttamente disponibile e in tal caso lo istanzia. Tale approccio è utilizzato da Firefox ed Opera. Se non è direttamente disponibile allora si tratta di IE. In tal caso l'oggetto in questione è disponibile attraverso un *ActiveX*. Il problema è che vi sono diverse versioni. Il ciclo *for* cerca di istanziare l'oggetto dal più recente al più datato catturando le eventuali eccezioni lanciate per la non disponibilità dell'oggetto. Ovviamente, se arriva alla fine del ciclo, vuol dire che *XMLHttpRequest* non è proprio disponibile per il browser in uso e viene notificato un avviso.
Ora che abbiamo la funzione che ci restituisce l'oggetto core di Ajax possiamo analizzare la gestione dei diversi tipi di response citati ad inizio articolo.

L'APPLICAZIONE D'ESEMPIO

Per esaminare i vari tipi di response che è possibile processare in un'applicazione Ajax-based, svilupperemo una semplicissima pagina web che ci permetterà di filtrare la lista dei dipendenti di una società fittizia. La scrematura avverrà attraverso una textbox che si aspetterà, in input, le iniziali del cognome degli impiegati da cercare. Ovviamente, le chiamate al server saranno di tipo asincrono. La parte server-side sarà costituita da un'unica pagina PHP che restituirà il response nei quattro formati citati (Plain Text, JSON, XML e HTML) a seconda di un parametro passato nella richiesta. Per chi non conosce PHP nessun problema, la parte server è così semplice che tradurre il codice PHP in Java, ASP, ASP.NET o qualsiasi altro linguaggio sarà un gioco da ragazzi. Per tale ragione il lato server non sarà esaminato in quest'articolo. La trovate, tuttavia, nel codice allegato munita dei dovuti commenti.
La **figura 1** mostra la pagina iniziale della nostra applicazione.
In pratica, quando viene caricata la pagina parte una richiesta asincrona al server per il recupero di tutti i dipendenti. Il risultato è, poi, renderizzato attraverso una semplice tabella HTML. La cosa non è molto complicata. Tuttavia la nostra applicazione ha la peculiarità, come vedremo, di produrre quella tabella per qualsiasi tipo di response ricevuto. I prossimi paragrafi, infatti, illustreranno come creare codice HTML per response di tipo Plain Text, HTML, JSON ed XML. Questo sarà utile per rendersi conto del codice JavaScript necessario

Cognome	Nome	Indirizzo	Salario	Dipartimento
Alparone	Domenico	Via I. Tubi, 38 - Reggio Calabria	34000.00	Edilizia
Catanese	Antonino	Via Felice Benlecco, 11 - Roma	30000.00	IT
Degaetano	Domenico	Via Albert Einstein, 81 - Reggio Calabria	40000.00	Amministrazione
Ferraro	Antonio	Via Antonio Sisala, 25 - Reggio Calabria	30000.00	IT
Giardinello	Domenico	Via Leonardo Da Vinci, 123 - Reggio Calabria	38000.00	Logistica
Lacava	Alessandro	Via Paco Rinajo, 69 - Milano	30000.00	IT
Lacava	Michelangelo	Via Maria Tiammacchi, 72 - Reggio Calabria	40000.00	Amministrazione
Morda'	Domenico	Via Beato Leccolano, 25 - Roma	30000.00	IT
Parisi	Leone	Via Nerume, 6 - Milano	32000.00	IT
Passaniti	Francesco	Via Nonle May, 96 - Milano	30000.00	IT
Petrolino	Dario	Via Francoise Pecorette, 48 - Reggio Calabria	40000.00	Amministrazione
Pitarelli	Luigi	Via Archi Cep, 69 - Bologna	32000.00	Edilizia
Scoglio	Antonino	Via Condotti, 31 - Reggio Calabria	40000.00	Amministrazione
Scoglio	Vittorio	Via Sigmund Freud, 77 - Reggio Calabria	35000.00	Trasporti

***Fig. 1:** Pagina web che otteniamo all'avvio dell'applicazione.*

a processare i response sopra citati. Come promesso, ovviamente, sarà possibile filtrare per cognome di impiegato. Se provate, ad esempio, ad inserire la lettera "L" nella textbox avrete che solo gli impiegati che iniziano per "L" saranno visualizzati. La **figura 2** mostra il risultato di tale ricerca.
Il nostro codice, in particolare, è formato da una singola pagina HTML, due file JavaScript, due pagine PHP lato server e lo script per la creazione del semplice database degli impiegati. Una pagina PHP contiene solo le funzioni di utilità per collegarsi al database e le credenziali relative. L'altra contiene il codice necessario per produrre il response nei vari for-

Cognome	Nome	Indirizzo	Salario	Dipartimento
Lacava	Alessandro	Via Paco Rinajo, 69 - Milano	30000.00	IT
Lacava	Michelangelo	Via Maria Tiammacchi, 72 - Reggio Calabria	40000.00	Amministrazione

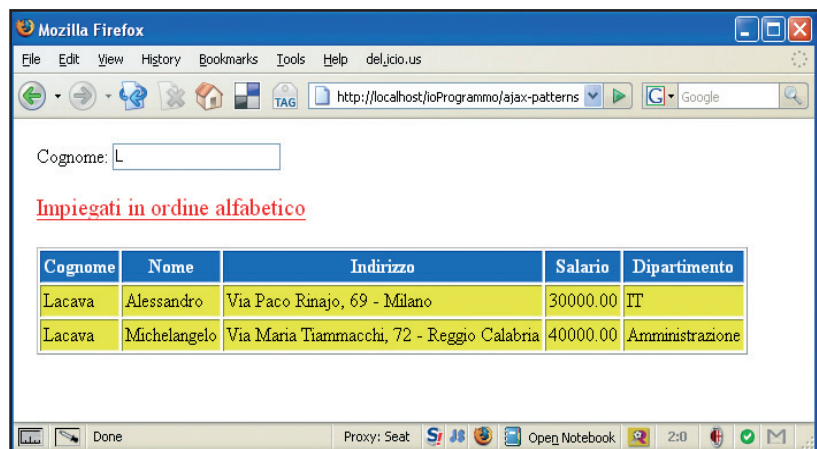
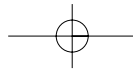



***Fig. 2:** Impiegati filtrati per lettera "L".*

NOTA

IL CODICE ALLEGATO

Gli esempi nell'articolo sono in Visual Basic; per chi fosse interessato, nel CD allegato è presente tutto il codice sorgente anche in C#.

mati elencati prima. Il *<body>* della singola pagina HTML, *employees.htm*, è il seguente:

```
<body onload="javascript:retrieveData('html');">

<div id="searchStringBlock">
        Cognome:
        <input type="text" name="searchString"
                              id="searchString"
onkeyup="javascript:retrieveData('html');"/>
</div>
<br />
<div id="dataRetrieved"></div>
</body>
```

Poca roba, vero? In realtà, tutta la logica per il recupero e la formattazione delle info concernenti gli impiegati è stata incapsulata in funzioni JavaScript che vedremo più avanti. Quel codice HTML, invece, produce solo la textbox di ricerca ed il <div>, *dataRetrieved*, atto a contenere la tabella risultante dalla chiamata Ajax. Sia al primo caricamento di *employees.htm* sia quando l'utente preme un tasto nella textbox, è invocata una funzione JavaScript per il recupero degli impiegati. La funzione in questione è *retrieveData* ed il parametro passato in ingresso indica il tipo di response desiderato:

```
function retrieveData(responseType)
{
var searchString =
  document.getElementById("searchString").value;
        var url =
              "http://localhost/ioProgrammo/ajax-
                patterns/employee_retriever.php";
        var xhr = createHttpRequest();
        xhr.open("get", url + "?output=" +
    responseType + "&lastName=" + searchString,
                                          true);
        //funzione di callback
        xhr.onreadystatechange = function()
        {
                processResponse(xhr,
                                   responseType);

        }
        //invia la richiesta
        xhr.send(null);
}
```

Per prima cosa, il codice precedente, recupera il contenuto della textbox. Ovviamente, se è la prima chiamata, esso sarà costituito da stringa vuota. Dopodiché prepara l'URL per la chiamata al server. In seguito ottiene un'istanza di *XMLHttpRequest* (XHR da ora in poi) utilizzando la funzione vista in precedenza. Notate che la funzione di callback, invocata ogni volta che lo stato della chiamata Ajax cambia, chiama *processResponse* passandogli l'oggetto XHR contenente il response del server ed il tipo di response da utilizzare. La chiamata al server avviene passandogli le iniziali del cognome inserite dall'utente (parametro *lastName*) ed il tipo di response da restituire (parametro *responseType*).
In particolare, *responseType* può assumere quattro possibili valori:

- *html*, per indicare al server che si desidera un response di tipo HTML, ossia una tabella HTML bella e pronta da schiaffare dentro il <div>.
- *xml*, per response di tipo XML. Inviando questo parametro alla pagina PHP, come response otteniamo un XML con la seguente struttura:

```
<?xml version='1.0' encoding='UTF-8'?>
<employees>

        <employee>
                <lastName>Lacava</lastName>
<firstName>Alessandro</firstName>
                <address>Via Paco Rinajo, 69 -
                                   Milano</address>
                <salary>30000.00</salary>
                <depName>IT</depName>
        </employee>

        <employee>
<lastName>Lacava</lastName>
<firstName>Michelangelo</firstName>
                <address>Via Maria
     Tiammacchi, 72 - Reggio Calabria</address>
                <salary>40000.00</salary>
<depName>Amministrazione</depName>
        </employee>
```

JSON IN BREVE

JSON sta per JavaScript Object Notation. Esso è semplicemente un formato leggero di interscambio dati che utilizza la notazione letterale di JavaScript per rappresentare oggetti ed array. Sostanzialmente, quindi, è un sottoinsieme di JavaScript. Non entrerò nel dettaglio di JSON dato che gli ci ho scritto un intero articolo su ioProgrammo N. 108 – Novembre 2006. In breve, un oggetto, in JSON, è rappresentato con la seguente sintassi:

```
{
"cognome" : "Lacava",
"nome" : "Alessandro",
"anni" : 31
}
```

Per rappresentare un array, invece, si usa la seguente sintassi:

```
[
"Java",
        "C#",
"JavaScript"
]
```

Ovviamente potete combinare le due cose per ottenere array di oggetti (come abbiamo fatto in quest'articolo) ed oggetti che contengono array tra le proprietà. Per interpretare codice JSON in JavaScript si può utilizzare la funzione eval. Esistono anche diverse librerie per mappare da linguaggi come PHP, Java e C# a JSON. Maggiori info li potete trovare sul sito ufficiale, all'indirizzo: http://www.json.org

030-037 29-06-2007 16:05 Pagina 33


```
        [...]
</employees>
```

- *json*, per indicare al server che si desidera un response di tipo JSON. Il response JSON avrà un formato analogo al seguente:

```
[
        {
          "lastName" : "Lacava",
          "firstName" : "Alessandro",
          "address" : "Via Paco Rinajo, 69 -
                                        Milano",
    "salary" : "30000.00",
          "depName" : "IT"
        },
        {
                "lastName" : "Lacava",
                "firstName" : "Michelangelo",
                "address" z: "Via Maria
              Tiammacchi, 72 - Reggio Calabria",
                "salary" : "40000.00",
                "depName" : "Amministrazione"
        },
        [...]
]
```

- *plain*, per response di tipo Plain Text, ossia testo puro. Un tipo di response siffatto non è stato preso molto in considerazione dal mondo Ajax. In realtà, come vedremo, quando si hanno problemi di banda, ecc. questo tipo di response può risultare la scelta migliore in termini di performance. Vi basterà guardare il seguente esempio e confrontarlo con i precedenti per rendervi conto di cosa sto parlando:

```
Lacava;Alessandro;Via Paco Rinajo, 69 -
Milano;30000.00;IT|Lacava;Michelangelo;Via Maria
                              Tiammacchi, 72 - Reggio
            Calabria;40000.00;Amministrazione[...]
```

Notare che, in quest'ultimo caso ogni impiegato è separato dal carattere pipe (|), mentre i vari campi del singolo impiegato sono separati da un punto e virgola (;).
Come già accennato la funzione atta al processing del response è *processResponse*:

```
function processResponse(xhr, responseType)
{
        //controlla lo stato della risposta
        if(xhr.readyState == 4 && xhr.status ==
                                                200)
        {
                var rp = new
          ResponseProcessor(xhr, "dataRetrieved",
```

```
                                        responseType);
                        rp.produceHtml();
        }
}
```

Quando l'attributo *readyState* dell'oggetto XHR assume il valore 4 e status 200 vuol dire che abbiamo ricevuto il response finale del server ed è andato tutto a buon fine. In questo caso si istanzia un oggetto di tipo *ResponseProcessor* e poi viene chiamato il suo metodo *produceHtml* che crea il codice HTML e lo inserisce all'interno del contenitore (il <div> *dataRetrieved*). La classe *Response Processor* è la più importante di tutta l'applicazione. Essa, infatti, ha la responsabilità di eseguire il parsing del server response, costruire l'HTML relativo ed inserirlo nel <div>. La dichiarazione di tale classe è la seguente:

```
function ResponseProcessor(xhr, container,
                                        responseType)
{
        this.EMPTY_RESPONSE = "Nessun
                                impiegato presente";
        this.xhr = xhr;
        this.container = container;
        this.responseType = responseType;
}
```

La proprietà EMPTY_RESPONSE rappresenta una costante. In pratica coincide con la stringa restituita dal server nel caso in cui non vengono trovati impiegati per il cognome indicato nella richiesta. La figura 3 illustra questo caso speciale.
I parametri passati a *ResponseProcessor* sono:

- *xhr*, ossia l'oggetto *XMLHttpRequest* contenente il response del server.
- *container*, l'id del <div> in cui andrà inserita la tabella HTML rappresentante la lista degli impiegati.
- *responseType*, il tipo di response ottenuto dal server. È in base a questo parametro che

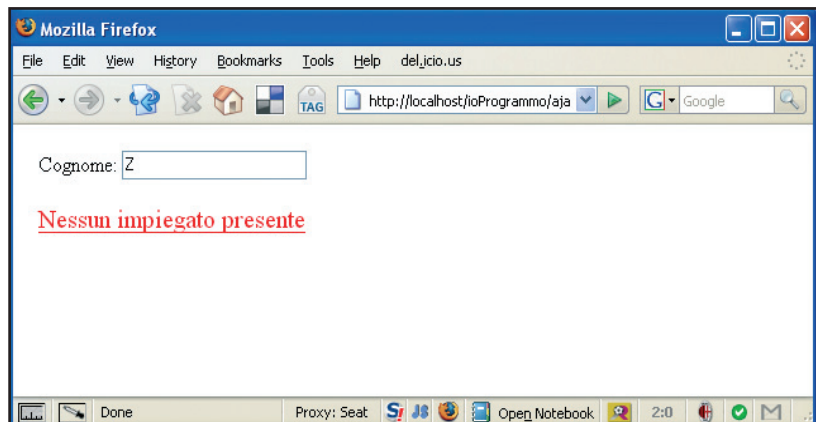
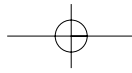
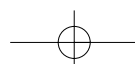



Fig. 3: Response relativo al caso in cui non viene trovato alcun impiegato

viene effettuato il parsing appropriato.

Come abbiamo visto, una volta istanziato un oggetto di tipo *ResponseProcessor*, viene chiamato il suo metodo *produceHtml* così codificato:

```
produceHtml : function()
{
        if(this.isEmptyResponse())
        {
                return;
        }
        var responseType =
                this.responseType.toLowerCase();
        switch (responseType)
        {
                case "plain":
                        this.processPlain();
                        break;
                case "xml":
                        this.processXml();
                        break;
                case "json":
                        this.processJson();
                        break;
                default:
                        this.processHtml();
        }
}
```

Esso, in pratica, controlla per prima cosa se il response è vuoto, ossia se non sono stati trovati dipendenti. In questo caso viene visualizza un messaggio adatto ed esce. Se il response non è vuoto, invece, viene invocato il metodo apposito a seconda del tipo di response. Esaminiamo i vari metodi separatamente dato che costituiscono il cuore dell'intero articolo. Partiamo dal default, ossia *processHtml*.

CASO HTML

Il metodo atto ad effettuare il parsing del response HTML è *processHtml*, eccone il codice:

```
processHtml : function()
{
        var html = this.xhr.responseText;
        this.fillContainer(html);
}
```

Come vedete, è molto semplice. Si recupera il contenuto del response tramite la proprietà responseText dell'oggetto XHR e si inserisce dentro il <div> tramite il metodo *fillContainer*:

```
fillContainer : function(content)
{
        var header = "<div style='color: #ff0000;
   font-size: 20px; text-decoration: underline;'>" +
                "Impiegati in ordine alfabetico" +
                "</div>" +
                "<br />";
document.getElementById(this.container).innerHTM
                        L = header + content;
}
```

Questo metodo, semplicemente, sbatte il codice HTML ricevuto in ingresso (più un header) all'interno del <div> atto a contenere la tabella degli impiegati. Il vantaggio principale del far formattare l'HTML lato server sta proprio nella semplicità del codice client associato. Infatti, non occorre fare nessun processing, basta prendere il codice ed inserirlo nel posto opportuno. Tra gli svantaggi spiccano una maggiore larghezza di banda occupata rispetto, ad esempio, al caso Plain Text e l'appesantimento del lato server dato che, oltre a recuperare le info, dovrà formattarle.

CASO PLAIN TEXT

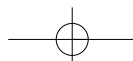
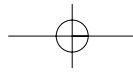
Il parsing del response di tipo Plain Text è eseguito dalla funzione *processPlain*:

```
processPlain : function()
{
        var response = this.xhr.responseText;
        var remappedEmployees = [];
        // Estrae i dipendenti dal response
                                        ricevuto
        var employees = response.split("|");
        var employeeFields;
```

AJAX IN BREVE

Si è parlato talmente tanto di Ajax in questo periodo che si è venuta a creare una gran confusione su cosa è ed a cosa serve. Ajax sta per Asynchronous JavaScript + XML. Il termine è stato coniato da Jesse James Garrett, presidente di Adaptive Path, in un articolo che potete trovare al seguente link: http://www.adaptivepath.com/publications/essays/archives/000385.php In sostanza, Ajax è un insieme di tecnologie utilizzate congiuntamente per sviluppare applicazioni, cosiddette, Web 2.0. Di quest'insieme di tecnologie fanno parte:

1. **(X)HTML e CSS, utilizzati per lo strato di presentazione.**
2. **DOM, per interagire dinamicamente col documento.**
3. **XML, come formato di interscambio dati.**
4. **XMLHttpRequest, per il recupero asincrono dei dati.**
5. **JavaScript, come "collante" delle precedenti.**

Vi è da dire, tuttavia, che non è obbligatorio utilizzare XML come formato di interscambio dati. In alcuni contesti è preferibile usare JSON, HTML o qualche altro formato customizzato (e.g. Plain Text) come visto in quest'articolo.

```
        for(var i = 0; i < employees.length; i++)
        {
                employeeFields =
                        employees[i].split(";");
                if(employeeFields.length != 5)
                {
                        throw new
                Error("Numero di campi errato");
                }
                var currentEmp = {
                    lastName : employeeFields[0],
                    firstName :
                                employeeFields[1],
                    address : employeeFields[2],
                    salary : employeeFields[3],
                    depName : employeeFields[4]
                };
remappedEmployees.push(currentEmp);
        }
        var html =
    this.buildHtmlFromObjArr(remappedEmployees);
        this.fillContainer(html);
}
```

Ricordiamo il formato del response Plain Text:

```
Lacava;Alessandro;Via Paco Rinajo, 69 -
 Milano;30000.00;IT|Lacava;Michelangelo;Via Maria
                         Tiammacchi, 72 - Reggio
               Calabria;40000.00;Amministrazione[...]
```

Il codice di *processPlain* splitta l'intero response per i vari pipe (|) ottenendo, così, un array di dipendenti. In seguito, splitta ogni dipendente per punto e virgola (;) ricavando i vari campi del singolo oggetto. Ciò che viene fatto, poi, è rimappare i dipendenti in un nuovo array di oggetti, dove ogni oggetto ha il seguente formato:

```
{
  lastName : "Lacava",
  firstName : "Alessandro",
  address : "Via Paco Rinajo, 69 - Milano",
  salary : "30000.00",
  depName : "IT"
}
```

Il vantaggio di questo "remapping" sta nel fatto che il response di tipo JSON si trova già in questo formato. Portando anche il response Plain Text in questa forma, possiamo riutilizzare lo stesso metodo per mettere l'array di oggetti in formato tabella HTML. Il metodo che ha quest'ingrato compito è *buildHtmlFromObjArr* che non esaminiamo per ragioni di spazio. Esso, tuttavia, è molto semplice, infatti, cicla sull'array di oggetti passato in ingresso e produce la tabella HTML rappresentante la lista di impiegati. Tale tabella è restituita in uscita dal metodo, come valore di ritorno. Tramite il solito *fillContainer*, poi, la tabella viene inserita nel <div>. Il vantaggio principale di questo tipo d'approccio sta nel risparmio di banda che si ottiene codificando il response nel formato Plain Text. Lo svantaggio primario è che tale formato non è molto user-friendly da un punto di vista "umano". Infatti, è molto più intuitivo leggere un response di tipo XML che non Plain Text.

CASO XML

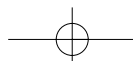
Il metodo *processXml* esegue il parsing del response di tipo XML:

```
processXml : function()
{
        var response = this.xhr.responseXML;
        var remappedEmployees = [];

        // Estrae i dipendenti dall'XML
                                  ricevuto
        var employees =
    response.getElementsByTagName('employee');
        for(var i = 0; i < employees.length; i++)
        {
                var currentEmp = {
                    lastName :
        this.getNodeValue(employees[i], 'lastName'),
                    firstName :
        this.getNodeValue(employees[i], 'firstName'),
```

AVVIO DELL'APPLICAZIONE

Per provare l'esempio di quest'articolo avete bisogno di PHP, un Web server e MySQL.
Il primo lo potete trovare qui: http://www.php.net/
Come Web server io ho utilizzato Apache HTTP Server che potete trovare all'indirizzo: http://httpd.apache.org/
Per quanto riguarda MySQL il sito di riferimento è il seguente: http://www.mysql.com/

Se utilizzate Windows e volete un'installazione molto semplice di PHP, Web server e MySQL vi consiglio di utilizzare EasyPHP. Esso installa, in un colpo solo, PHP, Apache Web server, MySQL e PHPMyAdmin. L'indirizzo per EasyPHP è il seguente: http://easyphp.org/

La creazione del database può essere fatta, semplicemente, utilizzando PHPMyAdmin. Una volta creato il DB potete utilizzare lo script che trovate nel codice allegato per la creazione ed il riempimento delle tabelle di esempio. Notate che dovete anche cambiare la variabile $dbName del file config.inc.php di modo che rifletta il nome che avete dato al database. Stesso dicasi per $dbHost, $dbUser e $dbPassword.
Una volta installato ed avviato il server, potete avviare l'applicazione utilizzando un URL simile al seguente: http://localhost/ajax-patterns/employees.htm
Questo supposto che avete impostato Apache HTTP Server in ascolto sulla porta di default (80) e messo il codice allegato sotto la directory ajax-patterns, a sua volta inserita sotto la root puntata da localhost.

```
                address :
          this.getNodeValue(employees[i], 'address'),

                salary :
          this.getNodeValue(employees[i], 'salary'),
                depName :
          this.getNodeValue(employees[i], 'depName')
                };

        remappedEmployees.push(currentEmp);
        }
        var html =
    this.buildHtmlFromObjArr(remappedEmployees);
        this.fillContainer(html);
}
```

Riportiamo, per comodità, il formato XML di cui compiere il parsing:

```
<?xml version='1.0' encoding='UTF-8'?>
<employees>
<employee>

  <lastName>Lacava</lastName>
  <firstName>Alessandro</firstName>

  <address>Via Paco Rinajo, 69 -
                              Milano</address>
  <salary>30000.00</salary>
  <depName>IT</depName>
</employee>
[...]
```

In pratica, quindi, è costituito da un insieme di nodi employee, dove ogni nodo rappresenta un impiegato. La seguente istruzione ottiene una lista di nodi che rappresenta tutti gli impiegati:

```
var employees =
        response.getElementsByTagName('employee');
```

Tale lista è, poi, scorsa in modo da eseguire il remapping in array d'oggetti per il motivo analogo al caso Plain Text. L'unica parte degna di nota è rappresentata dalle chiamate al metodo *getNodeValue* così implementato:

```
getNodeValue : function(parentElem, elem)

{
        return
parentElem.getElementsByTagName(elem)[0].firstC
                                    hild.nodeValue;
}
```

Tale metodo prende in ingresso due parametri:

- *parentElem*: ossia l'elemento padre da cui estrarre il contenuto di un nodo figlio
- *elem*: il nodo, figlio di *parentElem*, di cui estrarre il contenuto.

Ad esempio, passando *employees[0]* come *parentElem* e "lastName" come *elem* otteniamo il cognome del primo impiegato della lista *employees*. L'inserimento del codice HTML avviene, alla fine, tramite il solito *fillContainer*. Il vantaggio principale di esporre i dati in formato XML lato server sta nell'ampia conoscenza e comoda leggibilità di questo formato. Lo svantaggio rispetto a Plain Text o JSON sta nel maggior spreco di banda. Inoltre, fare il parsing di response di tipo JSON è molto più semplice rispetto ad XML.

CASO JSON

Il metodo processJson è molto stringato come potete osservare:

```
processJson : function()
{
        var employees =
                    eval(this.xhr.responseText);

        var html =
            this.buildHtmlFromObjArr(employees);
        this.fillContainer(html);
}
```

Ricordiamo di seguito il formato JSON per gli impiegati:

```
[
  {
```

DICHIARAZIONE DI UNA CLASSE IN JAVASCRIPT

Senza entrare troppo nel dettaglio per creare una classe in JavaScript basta seguire il seguente paradigma:

```
// Costruttore
function NomeClasse(param1, param2, param3)
{
   this.param1 = param1;
   this.param2 = param2;
   this.param3 = param3;
}

// Metodi
NomeClasse.prototype =
{
   metodo1 : function()
   {
      // corpo metodo1
   },
   metodo2 : function(parametro)
   {
      // corpo metodo2
   }
};
```

La prima corrisponde, a grandi linee, alla definizione di un costruttore. Il secondo blocco, invece, serve a definire i vari metodi.

```
    "lastName" : "Lacava",

    "firstName" : "Alessandro",

    "address" : "Via Paco Rinajo, 69 - Milano",

    "salary" : "30000.00",
    "depName" : "IT"
  },
  [...]
]
```

In pratica, basta una singola istruzione per effettuare il parsing di testo in formato JSON. Tale istruzione è eval. Essa è una sorta d'interprete JavaScript, infatti esegue qualsiasi statement e non solo JSON. Nel nostro caso converte il response del server in un array di oggetti JavaScript.
La costruzione del codice HTML ed il suo inserimento nel <div> avviene sempre attraverso *buildHtmlFromObjArr* e *fillContainer*, rispettivamente.
I vantaggi dell'approccio JSON sono, innanzi tutto, la maggior semplicità di parsing e la leggerezza, pur rimanendo molto più leggibile rispetto al Plain Text. Lo svantaggio primario, rispetto ad XML, sta nel fatto che JSON non è molto conosciuto ed accettato quanto XML, anche se sta ricevendo consensi sempre più ampi perfino da colossi dell'informatica come Yahoo!. Quest'ultima, infatti, offre, tra i suoi servizi web, la possibilità di ottenere response in formato JSON.
Così facendo è possibile interrogare i servizi utilizzando chiamate asincrone anche verso domini diversi da quello di appartenenza. Non mi dilungo su questa tecnica, poiché ci ho scritto un intero articolo, dal titolo "JavaScript cross-domain", pubblicato su ioProgrammo N. 114 - Maggio 2007. Se vi siete persi il numero in questione, ricordate che è possibile ordinarlo come arretrato.

CONCLUSIONI

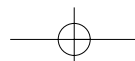
In quest'articolo abbiamo visto come gestire i tipi di response più utilizzati in ambito Ajax, ossia: HTML, XML e JSON. Abbiamo analizzato, inoltre, come gestire un response di tipo Plain Text da noi ideato.
Ovviamente è possibile migliorare l'applicazione sviluppata. Ad esempio, sarebbe opportuno gestire meglio gli eventuali errori tra client e server di modo da renderla più robusta. Inoltre, se lo ritenete opportuno, potete utilizzare un framework JavaScript per la gestione delle chiamate Ajax.
Ve ne sono svariati, tra i quali il valido Prototype (http://www.prototypejs.org/) a cui ho dedicato un intero articolo pubblicato su ioProgrammo N. 113 - Aprile 2007.
In ogni caso la scelta di un formato piuttosto che di un altro dipende da una miriade di fatgtori alcuni dei quali sono stati analizzati in questo articolo. Il primo passo come sempre è fare una buona analisi dei requisiti dell'applicazione che stiamo sviluppando. fatto questo possiamo optare se scegliere il buon vecchio "XML" oppure direzionarci su un tipo di response alternativo.

Alessandro Lacava

TIPO	VANTAGGIO PRINCIPALE	SVANTAGGIO PRINCIPALE
Plain Text	Il formato più leggero in assoluto, quindi occupa pochissima banda	Poco leggibile
XML	Molto leggibile ed ampiamente conosciuto	Un po' troppo verboso da un punto di vista della banda occupata
JSON	Semplicità di parsing e leggerezza	Non molto conosciuto
HTML	Non bisogna fare alcun processing lato client per costruire l'HTML da visualizzare	Occupa maggior banda rispetto al Plain Text ed aggiunge al server l'onere di formattare i dati

Abbiamo visto che i vari tipi di response presentano vantaggi e svantaggi. Ecco una tabella comparativa che li riassume.

VEDERE I TIPI DI RESPONSE

Abbiamo visto che retrieveData accetta un parametro che indica il tipo di response desiderato. Per com'è implementato ResponseProcessor, tuttavia, l'utente finale vedrà sempre e solo la tabella HTML prodotta. Da un punto di vista dello sviluppatore, ad ogni modo, può essere interessante vedere i vari tipi di response. A tal proposito potete puntare il vostro browser al seguente URL e cambiare il parametro output per modificare il tipo di response. I valori che è possibile dare ad output, sono: html, xml, json e plain. Ad esempio, il seguente URL produce un output di tipo XML: http://localhost/ajax-patterns/employee_retriever.php?output=xml Per filtrare per cognome potete aggiungere il parametro lastName. Il seguente esempio visualizza un response di tipo JSON per i soli impiegati che iniziano per "LA": http://localhost/ioProgrammo/ajax-patterns/employee_retriever.php?output=json&lastName=LA Ovviamente la ricerca è case-insensitive.

CREIAMO UN ASP.NET PERSONALIZZATO

UNA DELLE MAGGIORI POSSIBILITÀ OFFERTE DA ASP.NET È QUELLA DI POTER ESSERE ESTESO SULLA BASE DELLE ESIGENZE DEI PROGRAMMATORI. È UN PO' COME SE DISPONESSIMO DI PICCOLI MATTONCINI CHE CI SERVONO PER COSTRUIRE COMPONENTI PIÙ GRANDI...

Con il rilascio della versione 2.0 del framework .Net di Microsoft, lo sviluppo di applicazioni di gestione dati è diventata una attività molto semplice. I nuovi controlli di gestione dei dati (tipo *GridView* e *DetailsView*) ci consentono di sviluppare applicazioni orientate ai dati senza dover scrivere (quasi) nemmeno una riga di codice.
In più, laddove sia necessario, possiamo estendere i controlli forniti dal sistema per costruirne di nuovi, in grado di soddisfare quelle esigenze che i controlli di base non riescono a gestire.
In particolare, in questo articolo, vedremo come costruire un nuovo controllo a partire dal controllo *BoundField* che viene utilizzato come componente dei controlli di accesso e gestione dati (come appunto *GridView* e *DetailsView*), per visualizzare il valore di un campo dati come testo.

- *PoweredCheckBoxBoundField* che è simile al controllo *CheckBoxField* ma, in modalità di visualizzazione, verrà visualizzata una immagine o un opportuno carattere se il valore del relativo campo è "vero"; altrimenti non verrà visualizzato nulla.
  Questa implementazione rende sicuramente più chiara l'interpretazione del valore del controllo dato che la versione di base, in modalità di visualizzazione, mostra un CheckBox disabilitato, il che rende abbastanza difficile capire quali sono i campi con valore "vero" (*CheckBox* selezionato) rispetto a quelli con valore "falso" (*CheckBox* non selezionato).

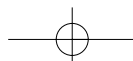
Prima di concentrarci nello sviluppo dei nuovi controlli, è bene definire delle linee guida che ci potranno essere utili anche in altre occasioni laddove avremo la necessità di estendere degli altri controlli al fine di renderli più consoni alle nostre esigenze; definiremo una sorta di paradigma che ci indicherà la strada per modificare i controlli esistenti.

"PROPERTIES & OVERRIDES"

Quando abbiamo la necessità si estendere un controllo di base, dobbiamo aggiungeva alcune caratteristiche che, altrimenti, non sarebbe in grado di soddisfare a pieno le nostre esigenze.
Alcuni esempi di caratteristiche avanzate potrebbero essere:

- la gestione della conferma della cancellazione di un record da un controllo di tipo *GridView* (nella versione base il record viene cancellato senza alcuna conferma);
- la possibilità di configurare un controllo di tipo *GridView* a partire da un documento XML;
- modificare il comportamento di un controllo *CheckBoxField* in modo che, in modalità di visualizzazione, verrà visualizzata una immagine o un opportuno carattere al posto di un controllo *CheckBox* disabilitato.

Potremmo altresì avere la necessità di costruire un controllo completamente nuovo come nel caso del controllo *PoweredDropDownBoundField* che andremo a sviluppare.
Il punto di partenza per lo sviluppo di un nuovo controllo è sicuramente l'individuazione del controllo di base da cui derivare la nuova classe (in questo ambito la parola "classe" e la parola "controllo" verranno utilizzati come sinonimi). Per definire controlli completamente nuovi potremmo partire dalle classi "madri" ovvero *Control* o *WebControl* che però forniscono di base solo le caratteristiche elementari di ogni altro controllo. Questo vorrebbe dire che da una parte avremo la massima flessibilità su cosa andremo a sviluppare ma dall'altra dovremo sviluppare praticamente tutto il codice del controllo.
Nella realtà, considerando anche la ricchezza di controlli forniti dal framework, di solito noi abbiamo una certa idea del controllo che ci serve, e riusciamo ad individuare anche un controllo di base che farebbe al nostro caso a meno di alcune caratteristiche che non vengono fornite nella versione

038-043 29-06-2007 16:12 Pagina 39

predefinita. In questa situazione, il controllo di base che più si avvicina alle nostre esigenze sarà quello da cui deriveremo il nostro nuovo controllo. Una volta individuata la classe di partenza, dobbiamo implementare le caratteristiche che ci servono. Di solito è sufficiente definire delle nuove proprietà della classe e gestire queste proprietà "sovrascrivendo" (ovvero costruendo un "override") alcuni metodi che vengono invocati durante il normale ciclo di vita del controllo; in questo modo avremo la possibilità di "iniettare" il nostro codice durante la fase di costruzione (o in qualsiasi altra fase) del controllo e, di conseguenza avremo la possibilità di produrre un controllo "potenziato".
Facciamo un esempio "qualitativo", ovvero a parole, del paradigma appena definito (analizzeremo due esempi completi nel seguito della trattazione): Supponiamo di voler estendere un controllo *GridView* in modo che l'utente venga avvisato, tramite un messaggio di conferma, quando decide di cancellare una riga.
La classe da cui andremo a derivare il nostro nuovo controllo, che potremmo chiamare *PoweredGridView*, sarà, ovviamente, il controllo *GridView*. Dato che dobbiamo gestire un messaggio di conferma, dovremo creare una proprietà di tipo stringa che contenga il messaggio che intendiamo mostrare all'utente quando cercherà di cancellare una riga della griglia.
Di seguito dovremo sovrascrivere il giusto metodo invocato durante il ciclo di vita del controllo. In questo caso dobbiamo associare del codice Javascript al link relativo al comando di cancellazione di un record, generato automaticamente dal controllo *GridView* durante la costruzione delle singole righe della griglia. Il metodo da sovrascrivere, allora, sarà il metodo *OnRowCreated*, all'interno del quale andremo ad inserire una semplice riga di codice che associa all'evento *OnClientClick* del controllo *LinkButton* relativo alla cancellazione, una chiamata alla funzione Javascript *confirm* che mostrerà all'utente il messaggio definito tramite la proprietà che abbiamo creato in precedenza.
In fase di esecuzione, non appena l'utente cercherà di cancellare una riga, verrà mostrata una finestra di dialogo che chiederà, appunto, la conferma della cancellazione.
Dopo aver illustrato qualitativamente il processo di creazione di nuovi controlli, passiamo all'analisi di due esempi concreti.

CONTROLLO POWEREDCHECKBOX

L'obiettivo è quello di creare un nuovo controllo che si comporti in modo simile al controllo *CheckBoxField* ma, in modalità di visualizzazione, quando il valore del relativo campo è "vero", vogliamo che venga visualizzata una immagine o un opportuno carattere al posto di un controllo *CheckBox* selezionato ma disabilitato; mentre quando il valore del relativo campo è "falso" non vogliamo mostrare nulla (al posto di un controllo CheckBox non selezionato e disabilitato).
Per la creazione di un nuovo controllo, dobbiamo aggiungere alla soluzione che stiamo sviluppando un nuovo progetto di tipo "Class Library"; chiaramente potremo scegliere di creare una soluzione a sé per ogni nuovo controllo in modo da poter poi riusare semplicemente la relativa dll in tutte le soluzioni che andremo a sviluppare.
La Figura 1 mostra il dialogo relativo alla creazione di un nuovo progetto in Visual Studio 2005.
Se applichiamo il paradigma "Properties & Overrides", per prima cosa dobbiamo individuare il controllo da cui derivare la nostra nuova classe. Sebbene possa sembrare ovvio ereditare da *CheckBoxField*, in realtà ci conviene derivare la nostra nuova classe dalla classe *BoundField* che è la classe padre di *CheckBoxField*. Questa scelta ci darà un po' più di libertà nella definizione del comportamento del nostro nuovo controllo.
A livello di codice, allora, abbiamo la seguente dichiarazione (codice contenuto nel file *PoweredCheckBoxBoundField.cs*)

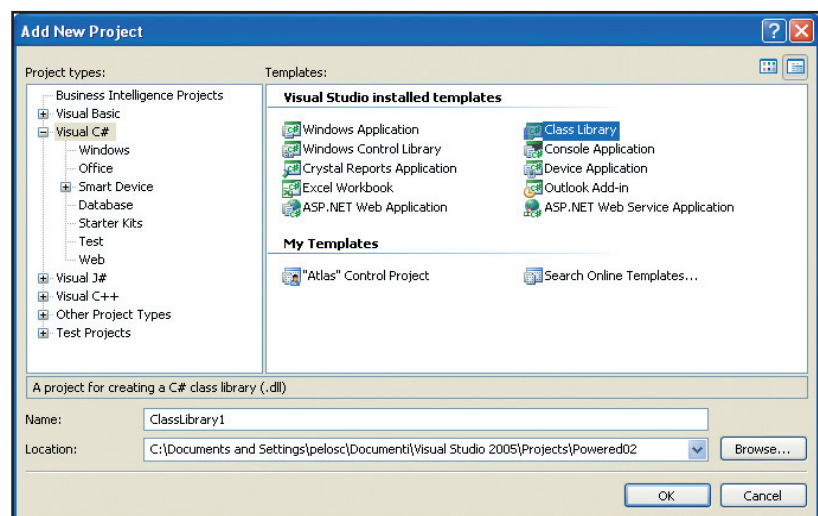
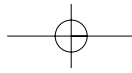
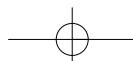



Figura 1: Per sviluppare un nuovo controllo creiamo un progetto di tipo "Class Library".

```
public class PoweredCheckBoxBoundField :
                                BoundField
{
...
}
```

Per la versione in linguaggio VB.Net fare riferimento al file *PoweredCheckBoxBoundField.vb*.

038-043 29-06-2007 16:12 Pagina 40

Il secondo passo è la definizione delle proprietà che vogliamo aggiungere al controllo. In questo caso abbiamo bisogno di una sola proprietà (*CheckedImagePath*) che rappresenta il percorso relativo (alla radice dell'applicazione) del file contenente l'immagine da mostrare all'utente; il relativo codice è:

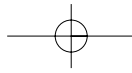
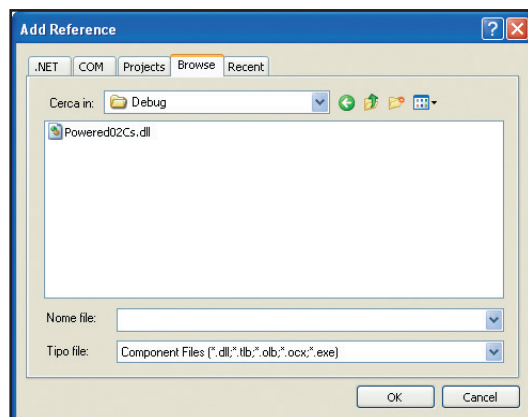
```
public string CheckedImagePath
{
 get
 {
  // Recupera il valore della proprietà dal ViewState
  object o = ViewState["CheckedImagePath"];
  // Se la proprietà è valorizzata ...
  if (o != null)
  {
   // Trasforma in stringa il valore della proprietà
   string strCheckedImageUrl = o.ToString();
   string sAppPath =
             HttpContext.Current.Server.MapPath("~/");
   // Definisce il percorso assoluto all'immagine
   strCheckedImageUrl = String.Concat(sAppPath,
                                  strCheckedImageUrl);
   // Verifica se il percorso fornito corrisponde ad un
                                          file fisico
   if (File.Exists(strCheckedImageUrl))
    return strCheckedImageUrl;
   else
    return "";
  }
  else
   return "";
 }
 set
 {
  // Memorizza il valore della proprietà nel ViewState
  ViewState["CheckedImagePath"] = value;
 }
}
```

Al fine di ottimizzare il comportamento del controllo, in fase di definizione del valore della proprietà *CheckedImagePath* (ramo "Get"), facciamo un test per verificare che il valore imposto alla proprietà sia relativo ad un file effettivamente presente nel sistema; in questo modo evitiamo di mostrare all'utente il segnaposto di "immagine non trovata" che il navigatore sostituirebbe nel caso di eventuale valore errato.

CODICE VB.NET

I progetti che analizzeremo nel corso della trattazione sono stati sviluppati sia in linguaggio C# che in linguaggio VB.Net. Per esigenze di spazio, nel corso della trattazione verrà illustrato solo il codice in linguaggio C#; per la corrispondente versione in linguaggio VB.Net fare riferimento ai file di supporto dell'articolo.

OVERRIDE INITIALIZECELL

Ultimo passo del processo di definizione del nuovo controllo è la sovrascrittura ("overrides") dei metodi invocati nel ciclo di vita del controllo.
Il primo metodo da codificare è *InitializeCell*, che è un metodo invocato da controlli tipo *GridView* o *DetailsView* per costruire la relativa struttura tabellare riga per riga. Il metodo viene utilizzato per inserire gli opportuni controlli all'interno dell'oggetto di tipo *TableCell* che rappresenta una singola cella dell'intera struttura tabellare. Il commento del codice è molto semplice; in primo luogo dobbiamo testare, attraverso il valore di input *cellType*, se la cella corrente appartiene ad una intestazione o ad un piè di pagina. In caso affermativo non dobbiamo far nulla dato che dobbiamo agire sulle righe dati, quindi invochiamo il relativo metodo della classe base.
In caso di "cella-dati", dobbiamo testare, attraverso il valore di input *rowState*, lo stato della riga: se è in stato di modifica o di inserimento, dobbiamo inserire un controllo di tipo *CheckBox*, altrimenti la riga è in stato di visualizzazione, quindi dobbiamo inserire un controllo di tipo *Label*. In entrambe i casi imponiamo l'identificativo del controllo (*CheckBox* o *Label*) pari al nome del campo dati a cui il controllo verrà associato (a breve capiremo il perché di questo "trucchetto"). In caso di "cella-dati" dobbiamo altresì aggiungere un gestore per l'evento dell'associazione dei dati ("binding") che ci consentirà di valorizzare il controllo in modo coerente con il valore del campo dati a cui verrà associato. Di seguito l'implementazione del metodo *InitializeCell*:

```
public override void InitializeCell(DataControlFieldCell
                                                cell,
    DataControlCellType cellType,
                        DataControlRowState rowState,
    int rowIndex)
{
 // Se la cella corrente è parte di una intestazione o
                                  di un piè di pagina ...
 // invoca il metodo di base
 if (cellType == DataControlCellType.Header ||
     cellType == DataControlCellType.Footer)
 {
```

```
 base.InitializeCell(cell, cellType, rowState,
                                           rowIndex);
 }
 // Se la cella corrente è parte di una riga di dati
 if (cellType == DataControlCellType.DataCell)
 {
  // Se la riga è in stato di "Edit" o di "Insert" ...
  // aggiunge un controllo CheckBox
  if ((rowState & DataControlRowState.Edit) != 0 ||
      (rowState & DataControlRowState.Insert) != 0)
  {
   CheckBox chk = new CheckBox();
   // Imponiamo l'identificativo del controllo
   // pari al nome del campo dati
   chk.ID = DataField;
   cell.Controls.Add(chk);
  }
  // altrimenti la riga è in stato di visualizzazione ...
  // aggiunge un controllo Label
  else
  {
   Label lbl = new Label();
   // Imponiamo l'identificativo del controllo
   // pari al nome del campo dati
   lbl.ID = DataField;
   cell.Controls.Add(lbl);
  }
  // Aggiunge un EventHandler per gestire l'evento di
                                        Data Binding
  cell.DataBinding += new
                 EventHandler(this.OnDataBindField);
 }
}
```

OVERRIDE ONDATABINDFIELD

Quando il controllo viene associato ad una sorgente dati, ovvero quando si effettua il "binding" dei dati, viene invocato il metodo *OnDataBindField* che dobbiamo andare a codificare per stabilire il giusto valore che deve assumere il nostro controllo.

```
protected override void OnDataBindField(object
                              sender, EventArgs e)
{
 // Cast del "sender" ad un controllo di tipo
                              DataControlFieldCell
 DataControlFieldCell cell =
                       (DataControlFieldCell)sender;
 // Ricaviamo il nome del campo dati
                                  dall'identificativo
 // del controllo aggiunto in fasei di inizializzazione
 string boundFieldName = cell.Controls[0].ID;
 // Ricaviamo il valore del campo
 string boundFieldValue =
```

```
                    GetBoundValue(cell.BindingContainer,
                                       boundFieldName);
 cell.HorizontalAlign = HorizontalAlign.Center;
 // Se il controllo presente nella cella corrente è di
                                          tipo Label ...
 if (cell.Controls[0] is Label)
 {
  // Se si deve visualizzare il valore ...
  if (boundFieldValue == "True")
  {
   // Se è definito il percorso verso una immagine ...
   if (CheckedImagePath != "")
   {
    // Crea una nuova immagine
    Image img = new Image();
    img.ImageUrl = CheckedImagePath;
    img.ImageAlign = ImageAlign.Middle;
    // Sostituisce la Label con l'immagine
    cell.Controls.Clear();
    cell.Controls.Add(img);
   }
   // altrimenti si visualizza un carattere
   else
   {
    Label lblCheck = new Label();
    lblCheck.Font.Size = FontUnit.Large;
    lblCheck.Font.Name = "webdings";
    lblCheck.EnableTheming = false;
    lblCheck.Text = "£";
    // Sostituisce la Label con il caratere
    cell.Controls.Clear();
    cell.Controls.Add(lblCheck);
   }
  }
 }
 // Se il controllo presente nella cella corrente è di
                                       tipo CheckBox ...
 else if (cell.Controls[0] is CheckBox)
 {
  // Se il valore del campo è "True" ...
  if (boundFieldValue == "True")
  {
  // Individua quindi seleziona il relativo controllo
                                               CheckBox
  CheckBox chk = (CheckBox)cell.Controls[0];
  chk.Checked = true;
  }
 }
}
```

Come prima cosa convertiamo il generico oggetto di input *sender* in un controllo di tipo *DataControlFieldCell* che rappresenta la cella, relativa al campo dati, in cui sono inseriti i controlli che dobbiamo valorizzare. Di seguito, attraverso l'identificativo del primo controllo presente nella cella, ricaviamo il nome del campo dati a cui si riferisce il nostro controllo; se facciamo un passo indietro ci ricordiamo che ave-

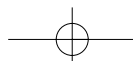
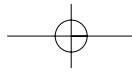
vamo imposto noi, in fase di inizializzazione, questa relazione tra l'identificativo del controllo presente nella cella ed il relativo nome del campo dati. Dal nome del campo dati ricaviamo il relativo valore attraverso la funzione *GetBoundValue* (che andremo ad analizzare a breve). Infine, in base al tipo di controllo presente nella cella corrente, ed in base al valore del campo,valorizziamo il controllo; in particolare possono verificarsi le seguenti situazioni: Il controllo presente nella cella corrente è di tipo *Label*, ovvero, da quanto stabilito in fase di inizializzazione, la relativa riga è in stato di visualizzazione; in questo caso se il valore del campo è "vero" (o "true" che dir si voglia), mostriamo l'immagine a cui fa riferimento il valore della proprietà *CheckedImagePath* se questa contiene un valore, altrimenti si visualizza un opportuno carattere (ovvero il carattere *webdings* £). Se il valore del campo è "falso" (ramo *else* nel codice), non mostriamo nulla. Se, invece, il controllo presente nella cella corrente è di tipo *CheckBox*, ovvero, da quanto stabilito in fase di inizializzazione, la relativa riga è in stato di modifica o di inserimento, se il valore del campo è "vero", dobbiamo individuare, quindi selezionare la *CheckBox*, altrimenti non dobbiamo far nulla dato che la *CheckBox* viene già visualizzata come non selezionata.

Figura 2: Un esempio di come apparirà la form

L'AUTORE

Oscar Peli è .NET Solution Architect ed amministratore dei dispositivi mobili presso il Comune di Ancona. Come consulente ha sviluppato presso l'Università degli Studi di Macerata un sistema di pubblicazione di contenuti per il supporto a progetti di ricerca (http://reti.unimc.it). Oscar è contattabile all'indirizzo: opeli@unimc.it.

FUNZIONE GETBOUNDVALUE

La funzione *GetBoundValue*, invocata dal metodo *OnDataBindField*, riceve in input sia il nome di un campo dati, sia un oggetto di tipo *BindingContainer* che contiene le informazioni relative ai dati che ci servono per valorizzare il nostro nuovo controllo. Per chiarire il concetto è bene focalizzare un esempio reale; nel caso in cui utilizzassimo il nostro nuovo controllo in una *GridView*, il *BindingContainer* sarebbe un oggetto di tipo *GridViewRow* che contiene le informazioni del record corrispondente alla riga in cui è inserito il nostro controllo. In particolare l'oggetto *BindingContainer* implementa l'interfaccia *IDataItemContainer* che, attraverso la proprietà *DataItem*, espone un oggetto di tipo *DataRowView* che rappresenta il record associato alla riga della *GridView*. Utilizzando il secondo parametro in input alla funzione come indice per l'oggetto *DataRowView* riusciamo a ricavare il valore del campo relativo. Il codice di seguito rappresenta la funzione *GetBoundValue*.

```
private string GetBoundValue(Control
                              controlContainer, string field)
{
 IDataItemContainer dic = controlContainer as
                                    IDataItemContainer;
 DataRowView drv = (DataRowView)dic.DataItem;
 return drv.Row[field].ToString();
}
```

EXTRACTVALUES FROMCELL

Per completare il nostro nuovo controllo, è necessario implementare un ultimo metodo, ovvero *ExtractValuesFromCell*. Questo metodo viene utilizzato per far si che i "binding container" tipo *GridView* or *DetailsView* possano estrarre il valore del controllo quando si trovano nello stato di modifica o di inserimento. Il codice del metodo è semplice; come prima cosa dobbiamo verificare se la cella corrente contiene un controllo di tipo *CheckBox*; in caso affermativo vuol dire che la riga corrente è in modalità di modifica o di inserimento (per come abbiamo strutturato il controllo) quindi dobbiamo semplicemente aggiornare le relativa voce nel parametro di input *dictionary* (che implementa l'interfaccia *IOrderedDictionary*).

```
public override void
  ExtractValuesFromCell(IOrderedDictionary dictionary,
    DataControlFieldCell cell, DataControlRowState
                       rowState, bool includeReadOnly)
{
 // Controlla che la cella corrente contenga un
                                          CheckBox
 // ovvero sia in modalità "edit" o "insert"
 if (cell.Controls[0] is CheckBox)
 {
```

038-043 29-06-2007 16:12 Pagina 43


```
    // Individua il controllo
    CheckBox chk = (CheckBox)cell.Controls[0];
    // Aggiorna la relativa voce del parametro di input dictionary
    if (dictionary.Contains(DataField))
     dictionary[DataField] = (chk.Checked ? "1" : "0");
    else
     dictionary.Add(DataField, (chk.Checked ? "1" : "0"));
  }
}
```

Sebbene questo metodo non entri esplicitamente nel discorso che abbiamo svolto finora sul ciclo di vita del controllo, è ugualmente fondamentale per il corretto funzionamento del controllo stesso. Se vogliamo cercare di contestualizzare l'uso del metodo, dobbiamo pensare ai meccanismi automatici che stanno dietro all'utilizzo di controlli del tipo *GridView*, associati, ad esempio, a controlli di tipo *SqlDataSource*. In particolari condizioni, ovvero quando il nome dei parametri utilizzati nelle stringhe di comando che il controllo *SqlDataSource* invia alla base dati, coincidono con i nomi dei campi delle relative tabelle che si vanno a modificare, il sistema riesce in modo automatico a passare i dati dai controlli dell'oggetto *Gridiew* ai parametri dell'oggetto *SqlDataSource*. Per effettuare questo passaggio automatico, il controllo *GridView*, interroga proprio un oggetto che implementa l'interfaccia *IOrderedDictionary* e che abbiamo opportunamente valorizzato per mezzo del metodo *ExtractValuesFromCell*. Relativamente al presente metodo è interessante analizzare anche l'implementazione del metodo in linguaggio VB.Net, dove possiamo constatare la leggera differenza di codifica dovuta alla mancanza, in VB.Net, di espressioni "veloci" per esprimere costrutti "if-then-else" semplici.

```
Public Overrides Sub ExtractValuesFromCell(_
   ByVal dictionary As IOrderedDictionary, _
   ByVal cell As DataControlFieldCell, _
   ByVal rowState As DataControlRowState, _
   ByVal includeReadOnly As Boolean)
 ' Controlla che la cella corrente contenga un CheckBox
 ' ovvero sia in modalità "edit" o "insert"
 If TypeOf (cell.Controls(0)) Is CheckBox Then
  ' Individua il controllo
  Dim chk As CheckBox = CType(cell.Controls(0), CheckBox)
  ' Aggiorna la relativa voce del parametro di input dictionary
  If (dictionary.Contains(DataField)) Then
   If chk.Checked Then
    dictionary(DataField) = "1"
   Else
    dictionary(DataField) = "0"
    End If
   Else
    If chk.Checked Then
     dictionary.Add(DataField, "1")
    Else
     dictionary.Add(DataField, "0")
    End If
   End If
  End If
End Sub
```

Con l'implementazione di questo ultimo metodo possiamo considerare concluso lo sviluppo del nuovo controllo *PoweredCheckBoxBoundField*; in un prossimo articolo mostreremo come creare un controllo stringa che diventa editable al clic del mouse, e come utilizzare questi controlli in una web form

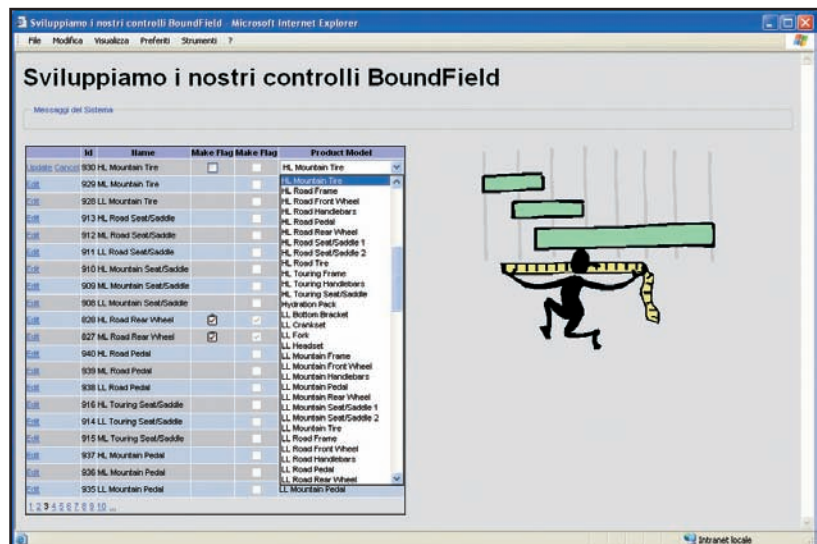
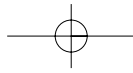
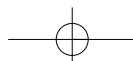



***Figura 3:** Il menu laterale compare dopo avere flaggato il chek box0*

CONCLUSIONI

Sviluppare un nuovo controllo vuol dire creare una libreria, ovvero una dll, in cui definiamo una nuova classe che potremo far derivare dalle classi base *Control* o *WebControl*. In questa ipotesi starà a noi sviluppare tutto il codice per supportare tutte le caratteristiche a noi necessarie. In realtà spesso capita però che i controlli a nostra disposizione siano "quasi" perfetti a meno di piccole modifiche. In questo caso se deriviamo la nuova classe da quella che più si avvicina al nostro ideale, possiamo sfruttare già tutte le caratteristiche del controllo base, ed a noi resta che aggiungere e/o modificare quelle caratteristiche che ci necessitano. La tecnica si rivela piuttosto potente. Ed ogno nuovo componente di venta base del successivo aumentando così di molto la p roduttività.

Oscar Peli

METTI AL RIPARO I TUOI DATI CON JAVA

PROTEGGERE I PROPRI DATI CONTRO OCCHI INDISCRETI, AI NOSTRI GIORNI ASSUME UNA RILEVANZA SENZA PRECEDENTI. JAVA METTE A DISPOSIZIONE MECCANISMI SEMPLICI PER LA SICUREZZA. ECCO LE BASI PER IMPARARE A CRITTOGRAFARE I DATI

Comprendere a fondo i meccanismi della crittografia ci permette di progettare sistemi tutti di una sicurezza estremamente elevata. Mettendo assieme i punti di forza di vari metodi crittografici è possibile implementare un protocollo sicuro per lo scambio di messaggi.
Nel precedente articolo abbiamo cominciato ad affrontare le tecniche e le problematiche legate alla crittografia. In questo articolo proseguiremo ulteriormente su questa strada sia approfondendo gli aspetti tecnico-implementativi sia di strutturazione quelli legati alla realizzazione di un'architettura robusta.
Prima di tutto quindi penso sia utile fare un brevissimo riassunto di quanto scritto nello scorso articolo.

CRITTOGRAFIA... MA QUALE E QUANDO?

Ribadiamo ancora una volta cosa si intende con il termine crittografia: *"l'insieme delle teorie e delle tecniche che permettono di cifrare un testo in chiaro o di decifrare un crittogramma"*. Come già detto tale definizione è molto generica e prescinde addirittura dal mezzo con cui essa viene effettuata. Naturalmente il nostro interesse ricade sulle moderne tecniche di crittografia, per intenderci quelle legate al mondo digitale e dei calcolatori elettronici.
Fatta tale premessa, possiamo dividere le tecniche crittografiche in due macro-categorie:

1. Crittografia a chiave segreta, o crittografia simmetrica
2. Crittografia a chiave pubblica, o crittografia asimmetrica

Crittografia a chiave segreta (o simmetrica)
Il sistema di crittografia a chiave segreta si basa sull'utilizzo di una unica chiave che serve sia per codificare il testo che per decodificarlo (da qui l'aggettivo simmetrico); in questo caso, tale chiave utilizzata deve essere nota sia al soggetto trasmittente che al destinatario. È quindi un vincolo imposto da questo sistema il fatto che sia il mittente sia il destinatario del messaggio condividano la medesima chiave e, poiché si devono scambiare la chiave di lettura in anticipo, è necessario che utilizzino dei sistemi di comunicazione sicuri, almeno per ciò che concerne il momento dello scambio della chiave. Un ulteriore vincolo imposto è la necessità di dover mantenere una diversa chiave per ogni interlocutore con cui desideri comunicare in maniera sicura.

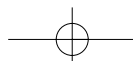
Crittografia a chiave pubblica (o asimmetrica)
La crittografia asimmetrica, o a chiave pubblica, si sviluppa negli anni Settanta e si basa su un sistema a doppia chiave: una chiave *privata* o *segreta*, conosciuta solo dal suo titolare, e una chiave *pubblica*, che è invece conoscibile da parte di chiunque , ad esempio in quanto pubblicata in un elenco consultabile tramite un opportuno servizio di Internet, e può quindi essere usata da qualsiasi altro utente.
La chiave pubblica serve ad applicare l'algoritmo matematico della cifratura ad un testo in chiaro; la chiave privata, invece, consente di decifrare il documento criptato con la chiave pubblica. Entrambe le chiavi sono diverse e non sono ricavabili l'una dall'altra. Questo tipo di crittografia ci libera quindi dalle limitazioni rilevate nella crittografia a chiave segreta.
Basandosi su quanto detto fin qui sembra che la crittografia asimmetrica sia la panacea di tutti i mali. In realtà le cose purtroppo non stanno proprio così e cerchiamo di capirne il perché.

PREGI E DIFETTI DEI VARI TIPI DI CRITTOGRAFIA

Vediamo nel concreto qual è il fine della nostra applicazione: scambiare messaggi con altre persone in maniera sicura anche quando non abbiamo la ga-

Conoscenze richieste
Conoscenze base di programmazione Java

Software
Java 2 Standard Edition SDK 1.5.0 o superiore, Internet Explorer o Mozilla, D-Bus 0.23 (solo per OS Linux)

Impegno
Tempo di realizzazione

ranzia che il canale su cui stiamo comunicando non lo sia. In questa situazione sicuramente l'adozione della crittografia a chiave pubblica può essere sotto certi aspetti estremamente utile, ma non risolve del tutto il problema; infatti quando si cripta in maniera asimmetrica ci si scontra con alcune limitazioni che, per i nostri intenti, risultano veramente pesanti.
Principalmente ne possiamo evidenziare subito due:

1. l'impiego degli algoritmi a chiave pubblica richiede un carico computazionale molto maggiore rispetto a quelli di crittografia simmetrica, quindi se ad esempio volessimo implementare un tipo di comunicazione real-time (tipo messanger) non potremmo garantire la prontezza nello scambio di messaggi.

2. Utilizzando la crittografia asimmetrica non è possibile criptare un testo di lunghezza maggiore di quella della coppia di chiavi stessa. In altre parole se ad esempio utilizzassimo una coppia di chiavi di lunghezza pari 2048 bit significherebbe poter codificare solamente 256 bytes (2048/8).

D'altro canto aumentare la lunghezza delle chiavi sarebbe nella pratica impossibile visti i tempi di codifica e decodifica richiesti (punto 1).
Vediamo proprio con un brevissimo stralcio di codice cosa comporta nella pratica quanto detto finora.

```
String text = "Nel mezzo del cammin di nostra vita
                                mi ritrovai per una selva
oscura ché la diritta via era smarrita."+
"Ahi quanto a dir qual era è cosa dura esta selva
     selvaggia e aspra e forte che nel pensier rinova la
                                                paura!"+
"Tant'è amara che poco è più morte; ma per trattar
     del ben ch'i' vi trovai, dirò de l'altre cose ch'i' v'ho
                                                scorte."+
"Io non so ben ridir com'i' v'intrai, tant'era pien di
 sonno a quel punto che la verace via abbandonai. Ma
                      poi ch'i' fui al piè d'un colle giunto,"+
"là dove terminava quella valle che m'avea di paura il
  cor compunto, guardai in alto, e vidi le sue spalle" +
"vestite già de' raggi del pianeta,che mena dritto
                                  altrui per ogne calle.";

String enText = "",deText ="";
Security.addProvider(new BouncyCastleProvider());
try {
            KeyPairGenerator gen =
```

AL SICURO.

Guadagni Copyright Applicativo Software House Licenze

SmartKey

SmartKey è il sistema hardware per la protezione degli applicativi. Un algoritmo proprietario e l'AES 128bit permettono un alto livello di protezione dalla copia abusiva. La certificazione IP67 la rende indistruttibile. **SmartKey** è driverless (DL) e automaticamente riconosciuta dai Sistemi Operativi Windows e Mac.

www.eutronsec.it
www.smartkey.eutronsec.it

EUTRONSEC INFOSECURITY

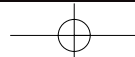
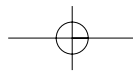
www.grafocom.it

```
KeyPairGenerator.getInstance("RSA", "BC");
gen.initialize(2048);
KeyPair kPair = gen.genKeyPair();
Cipher cipher =
Cipher.getInstance("RSA/ECB/PKCS1Padding","BC");
byte[] cipheredText = null;
try {
    cipher.init(Cipher.ENCRYPT_MODE,
        kPair.getPublic());
    cipheredText =
        cipher.doFinal(text.getBytes());
    enText = new
        String(Base64.encode(cipheredText));
    cipher.init(Cipher.DECRYPT_MODE,
        kPair.getPrivate());
    deText = new
```

***Figura 1:** procedura di crittazione mediante l'utilizzo di cipher blocks*

```
        String(cipher.doFinal(Base64.decode(enText)));
} catch (InvalidKeyException e) {...}
.....
}
```

Alla stringa text aggiungiamo di volta in volta una terzina della Divina Commedia lasciando inalterato tutto il codice sottostante. Sicuramente noterete che arrivati al punto riportato sopra verrà lanciata un'eccezione come questa:

```
Exception in thread "main"
    java.lang.ArrayIndexOutOfBoundsException: too
much
data for RSA block at
org.bouncycastle.jce.provider.JCERSACipher.engineDo
    Final(Unknown Source) at
    javax.crypto.Cipher.doFinal(DashoA12275) at
    test.Test01.main(Test01.java:45)
```

***Figura 2:** meccanismo di criptazione che utilizza sia la crittografia simmetrica che quella asimmetrica.*

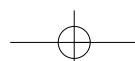
Naturalmente potremmo pensare di dividere il plain text in ingresso all'algoritmo di cifratura in nblocchi compatibili con la lunghezza della coppia di chiavi ma, dal punto di vista computazionale, sarebbe come aumentare la lunghezza della coppia stessa. Tale procedura è sintetizzata in figura 1.

A questo punto, come sovente accade nell'ingegneria software, ci ritroviamo ad essere costretti a ripensare nuovamente ad una soluzione che precedente ritenevamo risolutiva. In questo caso dobbiamo infatti riprendere in considerazione la crittografia simmetrica, che avevamo invece inizialmente scartato, poiché non è soggetta ai limiti posseduta da quella asimmetrica. Più precisamente dobbiamo porci il non facile obbiettivo di unire gli aspetti positivi di entrambi i sistemi di crittografia, facendo in modo che quelli negativi non si riflettano sul funzionamento generale della nostra applicazione. Ricapitolando quanto detto finora, possiamo riassumere la situazione come segue:

1. la crittografia simmetrica è computazionalmente efficiente e può quindi agevolmente criptare dati di qualsiasi lunghezza. D'altra parte tale tecnica presuppone che i due interlocutori siano a conoscenza della medesima chiave segreta.
2. la crittografia asimmetrica non ha bisogno invece che ambo gli interlocutori condividano la medesima chiave, infatti il mittente cripterà il messaggio con la chiave pubblica del destinatario, che a sua volta potrà decodificarlo tramite la propria chiave privata. Di contro, nella pratica, utilizzando la crittografia a chiave pubblica possiamo criptare solo messaggi di limitata lunghezza.

LE CHIAVI DI SESSIONE

Sarebbe quindi l'ideale riuscire ad utilizzare entrambi i tipi di crittografia in modo tale da sfruttare solo gli aspetti positivi di ciascuna di esse e, allo stesso tempo, superare quelli negativi.
L'idea è molto più semplice di quanto non si pos-

044-050 29-06-2007 17:43 Pagina 47

sa immaginare. Uno dei primi programmi a sfruttare il concetto che andremo ad illustrare fu proprio un'applicazione chiamata PGP (Pretty Good Privacy).
La **figura 2** rappresenta bene il meccanismo di criptazione che implementeremo. Abbiamo detto che non è nella pratica possibile criptare un lungo messaggio per mezzo della crittografia asimmetrica; allo stesso tempo non è sicuro scambiare in chiaro una chiave per la crittografia simmetrica. A questo punto la soluzione nasce in maniera naturale semplicemente dalla precedente analisi: perché non criptare la stessa chiave simmetrica (o segreta) con una chiave pubblica (cioè con la crittografia asimmetrica)? Uno scenario d'uso può aiutarci a chiarire le idee, al solito i nostri due interlocutori saranno Anna (A) e Bob (B)

1 A decide di mandare un messaggio criptato a B, quindi gli chiede la sua chiave pubblica che chiameremo Kpubb (se già non la possiede)
2 una volta avuta la chiave pubblica di B, A genera una chiave segreta (cioè una chiave per la crittografia simmetrica) “al volo”, ossia una chiave generata sul momento e che sarà utilizzata solo per quel messaggio e che poi può (anzi deve) venire cancellata (da qui anche il nome di “chiave di sessione”). Tale chiave verrà nominata con Ks.
3 A cripta il messaggio con un algoritmo di crittografia simmetrica utilizzando Ks. Ossia, utilizzando una notazione comune nel mondo della crittografia, se Mp era il messaggio in chiaro (plain-message), si ottiene il messaggio criptato Mc=f(Mp,Ks) dove f rappresenta l'algoritmo di criptazione simmetrica utilizzato.
4 A cripta anche Ks con la chiave pubblica di B, ottenendo così Kcs = g(Ks,Kpubb) dove g è l'algoritmo di criptazione asimmetrica utilizzato.
5 A spedisce a B la coppia (Mc,Kcs), ossia il messaggio criptato e la chiave segreta a sua volta criptata

Ora che Anna ha spedito il proprio messaggio a Bob, vediamo cosa quest'ultimo possa fare con i dati appena ricevuti.

1 B prende Kcs e ottiene Ks mediante $Ks=g^{-1}(Kcs, Kprib)$ dove Kprib è la chiave privata di B. E' importante notare ancora una volta che B è l'unico ed il solo a possedere la propria chiave privata e quindi l'unico ed il solo che possa recuperare Ks. Infatti neanche A potrebbe farlo (anche se è stata proprio lei a generala) poiché possiede esclusivamente la chiave pubblica di B ma non quella privata.

AL **SICURO.**

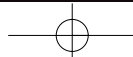
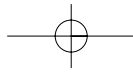
Software Portability Flash memory Autorun Multi partition

www.grafocom.it

PicoDisk 4CD

PicoDisk 4CD è la chiave di memoria USB 2.0 Hi Speed dedicata alle software house che intendano rendere utilizzabili le proprie applicazioni direttamente da una chiave USB.
La memoria flash del dispositivo può infatti essere partizionata in 4 differenti tipologie di unità logiche più un'area nascosta, che verranno combinate dalla software house in base alle esigenze della propria applicazione, arrivando a supportare la coesistenza su un unico dispositivo di un massimo di 4 partizioni più l'eventuale hidden area.
PicoDisk 4CD può essere un semplice Mass Storage opzionabilmente reso accessibile in sola lettura, un CD-ROM o entrambi, e può avere un'area di memoria completamente crittografata.

www.eutronsec.it
www.picodisk.com

EUTRONSEC INFOSECURITY

2 una volta ottenuta Ks, B riesce tranquillamente a decifrare l'intero messaggio semplicemente applicando $Mp=f^{-1}(Mc, Ks)$ si noti come la notazione con l'esponente -1 indichi la fase di decifratura dell'algoritmo corrispondente.

ANALISI DEL TERZO INCOMODO

Ora mettiamo alla prova il nostro meccanismo ipotizzando cosa un male intenzionato possa combinare per sabotare la comunicazione fra Anna e Bob. Il terzo incomodo lo chiameremo Charlie (C) e ci porremo nelle condizioni peggiori, ossia supporremo che Charlie possa rimanere in ascolto di tutto ciò che Anna e Bob si sono scambiati. Se anche C fosse riuscito a “sniffare” tutto ciò che Anna e Bob si sono scambiati, ossia (Kpubb,Kcs,Mcs), non potrebbe comunque decifrare nulla: infatti Mcs non è decifrabile se non possedendo Ks; a sua volta Ks non è ottenibile da Kcs se non possedendo Kprib e l'unico a possedere quest'ultima è proprio Bob.

Figura 3: applicazione prima che venga lanciata la criptazione

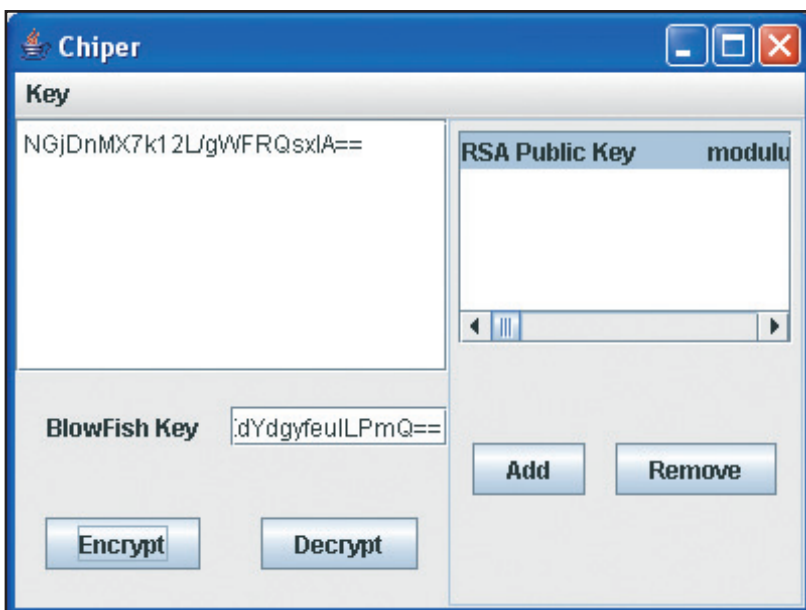



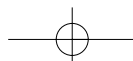
Figura 4: applicazione dopo che èstata lanciata la criptazione

DALLA TEORIA ALLA PRATICA

Mettiamo quindi in pratica quanto detto fin'ora implementando un'applicazione in Java. Prima di tutto fissiamo gli algoritmi che vogliamo utilizzare: per la crittografia asimmetrica verrà impiegato il classico RSA, mentre per quella simmetrica impiegheremo BlowFish (vedi BOX). Le figure 3 e 4 mostrano come si presenta la GUI sviluppata: a destra abbiamo una lista di chiavi pubbliche a nostra disposizione. I bottoni Add e Remove ci permettono aggiungere e rimuovere le chiavi ricevute e serializzate secondo lo standard opportuno (vedi articolo precedente). Ad esempio vediamo il codice sottostante il bottone Add:

```
...if (e.getSource() == jButtonAdd) {
if(publicKeyChooser.showOpenDialog(this) !=
                    JFileChooser.APPROVE_OPTION)
return;
File file = publicKeyChooser.getSelectedFile();
PublicKeyHandler kHandler = new
                               PublicKeyHandler();
try {
        kHandler.load(file.getAbsolutePath(),
                                          "RSA");
((DefaultComboBoxModel)jListKey.getModel()).addEle
                          ment(kHandler.getKey());
} catch (InvalidKeySpecException e1) {
        JOptionPane.showMessageDialog(this,
                     e1.getMessage(), "error",
                JOptionPane.ERROR_MESSAGE);
} catch (IOException e1) {
        JOptionPane.showMessageDialog(this,
                     e1.getMessage(), "error",
                JOptionPane.ERROR_MESSAGE);
}
}
```

Ricordo che le swing di Java sono state progettate secondo il pattern MVC ed è quindi necessario accedere al Model per aggiungere un elemento alla lista e cambiare di conseguenza lo stato della View.
La classe *PublicKeyHandler* è stata sviluppata nell'articolo precedente e ci permette di instanziare una PubblicKey a partire da uno stream di bytes, nel nostro caso un file. Quindi una volta caricata almeno una chia-

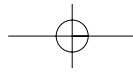
044-050 29-06-2007 17:43 Pagina 49

ve pubblica sarà possibile effettuare la procedura di criptazione precedentemente analizzata. Dalla figura 3 notiamo la presenza di un campo chiamato password. Essa costituirà il seme per la generazione della secret key da utilizzare con l'algoritmo BlowFish; in realtà, come si può anche evincere dai passi necessari alla fase di criptazione, basterebbe generare direttamente una chiave completamente casuale e nulla cambierebbe ai fini della procedura. La scelta di aggiungere un seme per la generazione della chiave segreta è stata fatta solamente "a scopi didattici", per rendere più espliciti i vari step. Vediamo quindi come nel concreto avviene tale criptazione:

```
private Cipher cipherRSA,cipherBlowFish;
private MessageDigest md;
private byte[] encryptedPwd=null,encryptedTxt=null;
.....
public MainFrame() {
try {
        cipherRSA =
  Cipher.getInstance("RSA/ECB/PKCS1Padding","BC");
        cipherBlowFish =
                Cipher.getInstance("blowfish","BC");
        md = MessageDigest.getInstance("MD5",
                                          "BC");
} catch (NoSuchAlgorithmException e) {...} }
```

BLOWFISH

in crittologia, Blowfish è un algoritmo a chiave simmetrica a blocchi, ideato nel 1993 da Bruce Schneier e implementato in molti software di crittografia. Sebbene a tutt'oggi non sia reperibile una crittanalisi di Blowfish, questo algoritmo sta suscitando nuovamente interesse se implementato con una maggior dimensione dei blocchi, come nel caso di AES o Twofish. Schneier progettò Blowfish per essere un algoritmo di utilizzo generale, utile a rimpiazzare l'allora decadente Data Encryption Standard (DES), e libero da problemi caratteristici di altri algoritmi. All'epoca molti altri sistemi di cifratura erano proprietari, coperti da brevetto o da segreti governativi. Schneier dichiarò: "Blowfish è libero da brevetti, e rimarrà tale in tutte le nazioni. L'algoritmo è di pubblico dominio, e può essere usato liberamente da chiunque". (tratto da Wikipedia)

Appena la GUI (MainFrame) viene creata inizializza due oggetti Chiper che implementano rispettivamente l'algoritmo RSA e blowfish. Quello che può generare un po' di stupore e viene da chiedersi è invece quale fosse la necessità di instanziare anche un oggetto per effettuare il digest di un testo visto che nei passi precedentemente illustrati non ve ne è traccia. Tale oggetto ci permetterà proprio di generare quella chiave segreta da dare in pasto a blowfish. Infatti, normalmente, una password avrebbe una lunghezza (espressa in

AL **SICURO.**

SmartPico

SmartPico è il dispositivo driverless che combina e fonde in un'unica chiave le caratteristiche di protezione per gli applicativi insite in SmartKey, insieme a quelle di praticità e trasporto dati tipiche di PicoDisk 4CD.
Grazie a queste particolarità è possibile installare un applicativo su **SmartPico** ed eseguirlo direttamente dalla chiave senza dover installare nulla sul PC. La memoria è partizionabile in più dischi, uno dei quali può avere la funzionalità CD e quindi supportare l'autorun.

www.eutronsec.it
www.smartpico.eutronsec.it

byte) troppo corta per essere efficacemente utilizzata direttamente come chiave segreta. Un algoritmo di hashing ci permette invece di ottenere una stringa di byte costante a partire da una stringa di byte qualsiasi. Nel nostro caso abbiamo scelto l'algoritmo MD5 che restituisce 128 bit in uscita, una lunghezza accettabile in termini di sicurezza per una chiave segreta. Il codice seguente riporta quindi la procedura di criptazione:

```
String newTxt = "";
try {
        PublicKey key =
            (PublicKey)jListKey.getSelectedValue();
        cipherRSA.init(Cipher.ENCRYPT_MODE, key);
        byte[] pwd =
            jTextFieldPassword.getText().getBytes
                (STRING_ENCODE);
        byte[] digest = md.digest(pwd); //128 bits
                = 16 byte
        SecretKeySpec spec = new
            SecretKeySpec(digest,"blowfish");
        cipherBlowFish.init(Cipher.ENCRYPT_MODE,
                spec);
        encryptedPwd = cipherRSA.doFinal(pwd);
        newTxt = new
    String(Base64.encode(digest),STRING_ENCODE);
        jTextFieldPassword.setText(newTxt);
        byte[] txt =
    jEditorPane.getText().getBytes(STRING_ENCODE);
        encryptedTxt =
        Base64.encode(cipherBlowFish.doFinal(txt));
        newTxt = new
            String(encryptedTxt,STRING_ENCODE);
        jEditorPane.setText(newTxt);
} catch (InvalidKeyException e) {
        JOptionPane.showMessageDialog(this,
            e.getLocalizedMessage(), "Error: ",
                JOptionPane.ERROR_MESSAGE);
} catch (IllegalBlockSizeException e) {
        JOptionPane.showMessageDialog(this,
            e.getLocalizedMessage(), "Error: ",
                JOptionPane.ERROR_MESSAGE);
} catch (BadPaddingException e) {
        JOptionPane.showMessageDialog(this,
            e.getLocalizedMessage(), "Error: ",
                JOptionPane.ERROR_MESSAGE);
} catch (UnsupportedEncodingException e) {
        JOptionPane.showMessageDialog(this,
            e.getLocalizedMessage(), "Error: ",
                JOptionPane.ERROR_MESSAGE);
}
```

Come avete modo di notare la prima cosa che si deve compiere è inizializzare i Chper in modalità encrypt e con la loro relativa chiave. In particolare, come già detto, quella per blowfish verrà creata a partire dalla password inserita dall'utente:

```
byte[] digest = md.digest(pwd); //128 bits = 16 byte
SecretKeySpec spec = new
        SecretKeySpec(digest,"blowfish");
cipherBlowFish.init(Cipher.ENCRYPT_MODE, spec);
```

Dopo di che avviene la criptazione vera e propria sia testo in chiaro che della password:

```
encryptedPwd = cipherRSA.doFinal(pwd);
........
byte[] txt =
jEditorPane.getText().getBytes(STRING_ENCODE);
encryptedTxt =
        Base64.encode(cipherBlowFish.doFinal(txt));
```

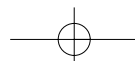
La parte rimanente del codice non fa altro che visualizzare quanto ottenuto; potete vederne un esempio in figura 4. A questo punto potremmo ad esempio fare un copia /incolla sulla nostra mail e spedire così il messaggio. Per motivi di spazio non verrà riportato il codice che effettua la decodifica, anche se, come potete immaginare, sarà praticamente simmetrico a quello appena riportato. Nel CD allegato alla rivista troverete comunque tutti i sorgenti che compongono la nostra applicazione.

PGP

È un programma che permette di usare autenticazione e privacy crittografica. Nelle sue varie versioni è probabilmente il crittosistema più usato al mondo. In Applied Cryptography, il crittografo Bruce Schneier lo ha descritto come il modo per arrivare "probabilmente il più vicino alla crittografia di livello militare"
PGP è stato originariamente sviluppato da Phil Zimmermann nel 1991. Il programma è l'insieme di diversi algoritmi di crittografia che garantiscono riservatezza e integrità dei dati (Md5, Idea, Rsa).
Con questo software è possibile firmare una E-mail lasciando il testo in chiaro, oppure cifrarla senza firmarla, o fare tutte e due le cose insieme. Supporta funzioni di gestione di un disco virtuale cifrato, PGP disk, garantisce lo scambio di dati in maniera sicura su un canale insicuro PGP vpn.

CONCLUSIONI

Questo articolo conclude l'argomento sui vari tipi di crittografia oggi disponibili. Inoltre questo nostro breve viaggio nel mondo della crittografia ci è anche servito a porci le problematiche legate ai contesti in cui essa viene impiegata. Come sempre avviene nella progettazione del software non esistono soluzioni perfette già preconfezionate, esistono bensì dei "mattoncini" che possono risolvere ben specifici aspetti e sarà onere di noi progettisti metterli assieme per costruire un edificio solido e robusto.

Alessandro Lacava

METTIAMO INSIEME SKYPE E GOOGLE

INTEGRIAMO DUE TRA LE PIÙ DIFFUSE TECNOLOGIE DEL MOMENTO: GOOGLE MAPS E SKYPE. REALIZZIAMO UNA SEMPLICE APPLICAZIONE PER VISUALIZZARE LA LISTA CONTATTI DI SKYPE SULLE MAPPE MESSE A DISPOSIZIONE DAL COLOSSO CALIFORNIANO

Skype è senza dubbio il programma di VoIP (Voice over IP) ed instant messaging più diffuso al mondo. Gli ultimi dati pubblicati lo confermano pienamente: oltre 500 milioni di downloads ed una community che supera 170 milioni di utenti.
Nel corso di questo articolo vedremo come integrare questo tool con uno dei servizi più utilizzati in rete: Google Maps, lo strumento messo a disposizione dal popolare motore di ricerca per la visualizzazione interattiva delle mappe geografiche. Poiché la soluzione che ci accingeremo a sviluppare, per quanto riguarda la parte client, sarà interamente implementata in Java, potremmo eseguire tale applicazioni su qualsiasi sistema operativo equipaggiato di un Java Runtime Environment con un client Skype in esecuzione.

GOOGLE MAPS

In questa prima parte dell'articolo focalizzeremo i nostri sforzi sul come integrare le mappe di Google all'interno della nostra applicazione. Google Maps è un servizio pensato per essere utilizzato unitamente ad un web browser connesso in rete.
Per interagire con le mappe lo sviluppatore ha a disposizione una API Javascript che permette di definirne la visualizzazione. Inoltre, sempre attraverso le API, è possibile specificare l'inserimento di immagini in determinati punti della mappa e definirne il comportamento a fronte di interazioni utente (es: mouse click). Google mette a disposizione questo servizio in maniera completamente gratuita, ma l'indirizzo della pagina che utilizza tali mappe deve essere registrato ed associato ad una specifica chiave. Quest'ultima deve essere utilizzata nelle chiamate alle funzioni Javascript definite nelle API (per maggiori informazioni visitare http://www.google.com/apis/maps/). Se il dominio da cui arriva la richiesta e la chiave non corrispondono non verrà visualizzata alcuna mappa ma uno specifico messaggio di errore. Visto che la nostra applicazione è di tipo standalone, in quanto faremo uso delle librerie grafiche AWT/Swing, saremo costretti ad utilizzare un piccolo workaround. L'applicazione, attraverso l'uso della libreria JDIC, integrerà il web browser di default del sistema e contatterà un server, che per comodità chiameremo SkypeMapServer, precedentemente registrato su Google Maps, che fungerà da ponte per visualizzare le mappe.
Visto che questo argomento è troppo vasto per essere trattato all'interno di questo articolo, non ci soffermeremo su come lo SkypeMapServer interroga le mappe di Google ma descriveremo soltanto come si aspetta di ricevere le informazioni relative agli utenti di Skype. La nostra applicazione dovrà "comunicare" tramite query-string le informazioni di ogni singolo utente con la seguente sintassi:

```
loc<index>=<city>%20<country>&han<index>=<u
    serName>;<displayName>;<alias>;<birthday>;<bu
        ddyStatus>;<onlineStatus>;<homepage>
```

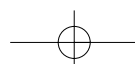
Le variabili all'interno della query-string saranno valorizzate con le seguenti informazioni utente:
∑
- **index**: rappresenta l'indice incrementale dell'utente Skype partendo da 0.
- **city**: la città dell'utente.
- **country**: è il codice della nazione in formato ISO-3166.
- **userName**: l'userName Skype dell'utente.
- **displayName**: il nome reale dell'utente.
- **alias**: l'eventuale alias.
- **birthday**: la data di nascita.
- **buddyStatus**: lo stato (In linea, SkypeMe, Assente, Non disponibile, Occupato).
- **onlineStatus**: nel caso l'utente sia on line deve assumere il valore "olsOnline".
- **homepage**: l'URL dell'eventuale pagina web associata.

SKYPE4JAVA

Una volta predisposta l'integrazione con le mappe non ci rimane che capire come interrogare Skype per conoscere le informazioni relative alla lista contatti. Skype4Java (https://developer.skype.com/wiki/Java_API) è una libreria open source che consente di "dialoga-

Conoscenze richieste
Conoscenze base di programmazione Java

Software
Java 2 Standard Edition SDK 1.5.0 o superiore, Internet Explorer o Mozilla, D-Bus 0.23 (solo per OS Linux)

Impegno

Tempo di realizzazione

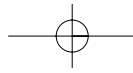
re" con i client Skype indipendentemente dal sistema operativo sul quale il tool viene eseguito. Questo progetto, che è nato dall'unione delle librerie Skype API for Java e JSA, può essere concettualmente suddiviso in due parti: la prima, contenente il codice Java su cui lavorerà lo sviluppatore finale e la seconda che comprende le librerie native richiamate tramite tecnologia JNI (Java Native Interface) per la comunicazione di basso livello con il client Skype.
Per ragioni di estendibilità, che approfondiremo successivamente, definiremo un'interfaccia che rappresenta un connettore ad un tool generico.

```
public interface Connector {
    public void connect() throws ConnectionException;
    public String getBaseUrl();
    public String getQueryString()throws
                                    ConnectionException;
}
```

NOTA

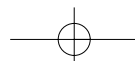
L'operatore! è detto fattoriale ed agisce così su un numero naturale: N! = N*(N-1)! e 0! = 1, quindi ad esempio 4! = 4*3!=4*3*2!=4*3*2*1! =4*3*2*1*0!=24

L'interfaccia espone tre metodi: connect() serve a stabilire la connessione al client, getBaseUrl() ritorna l'URL dello SkypeMapServer, getQueryString() interroga il client per ricevere la query-string contenente la lista contatti formattata secondo la sintassi descritta nel paragrafo precedente.
A questo punto non ci resta che definire l'implementazione dell'interfaccia che avrà il ruolo di "comunicare" con il client Skype.

```
public class SkypeConnector implements Connector {
    public void connect() throws ConnectionException
                                                  {
        try {
            Skype.isRunning();
        } catch (SkypeException ex) {
            throw new
                  ConnectionException(ex.getMessage());
        }
    }

    public String getBaseUrl() {
        return "http://www.jaco.it/maps/";
    }
    public String getQueryString() throws
                                ConnectionException {
        StringBuilder qString = new StringBuilder("?");
        try {
            ContactList contactList =
                                  Skype.getContactList();
            Friend[] friends =
                              contactList.getAllFriends();
            int num = 0;
            String pamvalue;
            String parameters;
            String city;
            String country;
            for (Friend friend: friends) {
                parameters = "loc";
                pamvalue = "";
                city = friend.getCity().trim();
                country = friend.getCountry().trim();
                if (city.length() > 0 && country.length()
                                                    > 1) {
                    pamvalue = city + "%20" +
                                 country.substring(0,2);
                } else {
                    if (city.length() > 0) {
                        pamvalue = city;
                    } else if (country.length() > 1) {
                        pamvalue =
                                 country.substring(0,2);
                    }
                }
                if (pamvalue.length() > 0) {
                    qString.append(parameters + num +
                                        "=" + pamvalue);
                    pamvalue = "";
                    parameters = "&han";
                    if (friend.getId() != "") {
                        pamvalue = friend.getId() + ";" +
                              friend.getDisplayName() + ";"
                            friend.getAbout + ";" +
                               friend.getBirthDay() + ";" +
                            friend.getOnlineStatus() +
                                                    ";" +
      (friend.getOnlineStatus().equals(User.Status.ONLINE)
                                     ? "olsOnline" : "") +
                             ";" +
                              friend.getHomePageAddress();
                        qString.append(parameters + num
                                + "=" + pamvalue + "&");
                    }
                    num++;
                }
            }
        } catch (SkypeException ex) {
            throw new
                  ConnectionException(ex.getMessage());
        }
        return qString.toString();
    }
}
```

Come possiamo notare, è il metodo getQueryString() il punto cruciale dell'intera classe. Al suo interno viene prima recuperato l'oggetto ContactList contenente un array di oggetti di tipo Friend e, per ognuno di essi, vengono recuperate le informazioni necessarie alla composizione della query-string da passare allo SkypeMapServer.

CREAZIONE DELLA GUI

L'ultimo step da realizzare riguarda l'interfaccia grafica

053-055 29-06-2007 16:17 Pagina 55

che permetterà all'utente di interagire con l'applicazione. Questo compito sarà affidato ad una semplice estensione di JFrame contenente una toolbar con all'interno un JButton per effettuare la connessione verso il client Skype e, nella zona centrale, sarà collocato un oggetto di tipo WebBrowser, fornito dalla libreria JDIC, in cui visualizzeremo la mappa.
In fase di iniazializzazione della classe verrà creata una HashMap statica contenente una chiave per identificare il connettore (che verrà utilizzata anche nella creazione del relativo bottone) e l'istanza di tipo Connector associata.

```
public static final HashMap<String, Connector>
                                        contactsToolMap;
    static {
        contactsToolMap = new HashMap(1);
        contactsToolMap.put("Skype", new
                                      SkypeConnector());
    }
```

Di seguito è mostrato il codice che rappresenta il fulcro di tale classe: l'implementazione del metodo actionPerformed() dell'interfaccia ActionListener, eseguito al verificarsi del click sul bottone posto nella toolbar.

```
public void actionPerformed(ActionEvent e) {
        Connector connector =
            contactsToolMap.get(e.getActionCommand());
        try {
            connector.connect();
        } catch (ConnectionException ex) {
            JOptionPane.showMessageDialog(this,
                "Verificare che " + e.getActionCommand() +
                    " sia in esecuzione: " +
                                        ex.getMessage(),
                    "Connection Error",
                        JOptionPane.ERROR_MESSAGE);
            return;
        }
        try {
            this.webBrowser.setURL(new
                    URL(connector.getBaseUrl() +
connector.getQueryString()));
        } catch (MalformedURLException ex) {
            JOptionPane.showMessageDialog(this,
                    ex.getMessage(), "Server Error",
JOptionPane.ERROR_MESSAGE);
            return;
        } catch (ConnectionException ex) {
            JOptionPane.showMessageDialog(this,
                    "Errore durantel'interrogazione " +
                    "della lista contatti:" +
                                        ex.getMessage(),
                    "Connection Error",
                        JOptionPane.ERROR_MESSAGE);
            return;
        }
            this.webBrowser.repaint();
    }
```

Il metodo estrae l'istanza di tipo Connector collegata al nome del comando legato all'azione ed invoca il metodo connect(). Se la connessione va a buon fine verrà creato l'URL da passare all'istanza di WebBrowser richiamando i metodi getBaseUrl() e getQueryString(). In Figura 1 è visualizzato uno screenshot dell'applicazione in esecuzione.

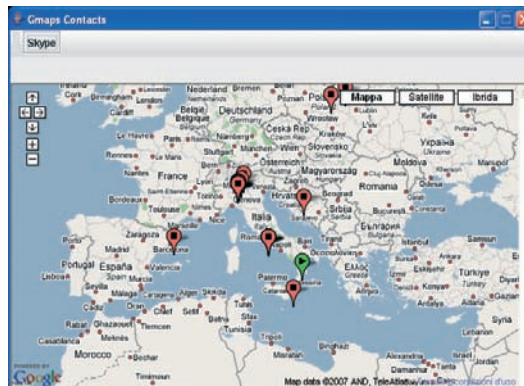



***Figura 1:** Ecco come la nostra applicazione visualizzerà i contatti di Skype*

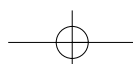
NON SOLO SKYPE

In questo articolo si è voluto trattare Skype per la sua massiccia diffusione. Sicuramente, però, esistono svariate applicazioni che interagiscono con una lista utenti contenente per ciascuno di essi le informazioni geografiche. Pertanto, abbiamo avuto l'accortezza di sviluppare il nostro codice in modo che sia facilmente estendibile ad altre applicazioni. Infatti, se si volesse aggiungerne una nuova basterebbe creare l'apposito Connector ed aggiungerne un'istanza nella HashMap statica creata nella classe ContactsMapFrame. Quest'ultima creerà in maniera trasparente l'apposito bottone nella toolbar e ne gestirà le chiamate verso lo SkypeMapServer.

CONCLUSIONI

In questo articolo abbiamo illustrato come attraverso l'integrazione di tre interessanti tecnologie sia possibile localizzare la lista contatti di Skype su una mappa geografica. L'applicazione implementata non vuole essere una soluzione completa, ma sicuramente può essere utilizzata come punto di partenza di un sistema più generale e sofisticato per la gestione geografica di una qualsiasi lista contatti. Infatti, con il modello utilizzato sarà possibile integrare altri strumenti senza il bisogno di riscrivere l'intera applicazione.

Fabrizio Fortino

056-061 29-06-2007 17:55 Pagina 56

DATABASE BERKLEY INTEGRATO IN JAVA

UNA PANORAMICA SU UNO DEI DB PIÙ PERFORMANTI DEL MOMENTO. PUÒ ESSERE UTILIZZATO IN MODO EMBEDDED ED È STATO REALIZZATO APPOSITAMENTE PER IL LINGUAGGIO DI SUN. IMPARIAMO COME SFRUTTARLO AL MASSIMO

In commercio ci sono database di vario tipo. Qualcuno sfrutta la potenza in lettura e scrittura, qualcuno le strutture *XML*, qualcuno il fatto di essere gratuito. Il DBBerkeley cerca di ottimizzare il passaggio dei dati tra applicazioni java e il db realizzando un database interamente realizzato in java. In questo modo si cerca di colmare quella distanza tecnologica che divide la memorizzazione dei dati dai linguaggi applicativi che li manipolano. L'interesse per questo db è stato tale da indurre Oracle ad acquisirlo e le sue caratteristiche ne spiegano il perché. Supporta db di grandi dimensioni, gestisce transazioni e thread multipli, sfrutta la semplicità per memorizzare i dati, usa indici e cache per aumentare le prestazioni, permette il recupero dei dati a seguito di crash di sistema e supporta connessioni con altre architetture (*JCA*) e gestioni delle estensioni (*JMX*). Da questa prima panoramica si può capire come sia forte il legame tra i dati e java; e come siano sfruttate le qualità di java. Negli esempi dell'articolo si mostreranno le caratteristiche del DBBerkeley per capire e realizzare applicazioni che leggono e scrivono su db e usano gli indici per migliorare le performance.

Fig. 1: DBBerkeley è un prodotto Oracle.

Conoscenze richieste
Basi di programmazione Java e di database

Software
JDK 1.4.2 o superiore, DB Berkeley

Impegno

Tempo di realizzazione

PRIMO APPROCCIO

Un inizio indolore al DBBerkeley è la classe *VerifyDB* che permette di verificare se il db è perfettamente funzionante. In altre parole possiamo iniziare da semplici utilizzatori del db, tralasciando, in un primo momento, gli aspetti implementativi. Quello che si richiede a ogni db è di poter inserire informazioni e poi leggere in un secondo momento: *VerifyDB* serve proprio a questo. Per inserire alcuni valori dobbiamo specificare l'operazione, dove si trova il db e quanti record mettere. L'istruzione di inserimento è la seguente:

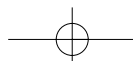
```
. insert 10 0
```

Mentre l'analoga istruzione per recuperare quello che abbiamo memorizzato è:

```
. retrieve
```

Le istruzioni sono quasi intuitive. La prima dichiara di inserire 10 elementi numerici a partire dal valore 0. La seconda dice di recuperarli. In comune hanno il punto iniziale che specifica la posizione del db. Potete cambiare questo valore adattandolo alle vostre esigenze oppure lasciarlo così, intendendo che il db verrà creato nella stessa directory dove c'è la directory classes che contiene i file compilati. Fisicamente, viene creato un file con estensione *jdb* che indica la presenza del nostro *Java Berkeley DB*.
Per chi ha familiarità con l'accesso ai db, può intuire che il codice di questo primo esempio si collega al db, esegue le transazioni per inserire i record e poi accede al db per leggere le informazioni. Tutti questi passaggi sono analizzati più in dettaglio con gli esempi seguenti.

L'AMBIENTE

Se si hanno esperienze con *jdbc*, una delle prime operazioni da eseguire è collegarsi al db usando, appunto, il driver *jdbc*. In questo caso,

invece, poiché il db è scritto interamente in Java, si eseguono operazioni differenti. L'esempio minimale che può essere usato per collegarsi al db o, come sono soliti dire gli inventori del db *Berkeley*, per aprire l'ambiente, è la classe *OpenEnviroment*.

```
// OpenEnviroment
DBEnviroment exampleDbEnv = new
                              DBEnviroment();
...
exampleDbEnv.setup(new File(DB_FILE), false);
...
 exampleDbEnv.close();
```

Queste sono le righe più importanti: creazione dell'oggetto che gestisce il db e impostazione dei parametri (DB_FILE è una costante dove mettere il nome del file che contiene il db). Si può dire che queste istruzioni sono le più generiche e basilari per aprire l'ambiente. In realtà, questa stessa classe verrà utilizzata in seguito per altri esempi e la seconda istruzione conterrà parametri diversi. Come si vede dalle righe precedenti, viene utilizzata la classe *DBEnviroment* per impostare i parametri.

```
// DBEnviroment
EnvironmentConfig myEnvConfig
   = new EnvironmentConfig();
myEnvConfig.setReadOnly(readOnly);
myEnvConfig.setAllowCreate(!readOnly);
myEnvConfig.setTransactional(!readOnly);
```

I nomi dei metodi per impostare i parametri sono abbastanza intuitivi. Servono a specificare se l'ambiente deve essere in sola lettura e se è transazionale. Inoltre, si può specificare se l'ambiente può essere creato o si deve usare un ambiente preesistente. Altri parametri possono essere modificati a run-time come la percentuale di *Java Virtual Machine* usata per il

```
key=0 data=1
key=1 data=2
key=2 data=3
key=3 data=4
key=4 data=5
key=5 data=6
key=6 data=7
key=7 data=8
key=8 data=9
key=9 data=10
key=10 data=11
key=11 data=12
key=12 data=13
key=13 data=14
key=14 data=15
key=15 data=16
key=16 data=17
key=17 data=18
key=18 data=19
key=19 data=20

Process finished with exit code 0
```

Fig. 2: Risultato della retrieve

db, o la cache usata, o la possibilità di tenere le informazioni in memoria senza registrarle fisicamente. In altre parole, per iniziare ad usare il db, vanno benissimo gli esempi riportati; se invece si cercano prestazioni eccellenti allora si possono modificare i parametri adattandoli ai casi specifici.
Dopo aver creato e caricato ambiente e db si passa alle operazioni che coinvolgono maggiormente i db: inserimento e lettura dei dati.

INSERIMENTO DATI

Inserire dati in un db realizzato in Java è simile all'uso delle *Map*, formate da una chiave e dal corrispondente valore. D'altronde, i dati nel db sono composti da una chiave, quella primaria, e dal resto delle informazioni che caratterizzano il dato. Anche il DBBerkeley lavora in questo modo. L'esempio che permette di inserire i dati nel db è composta da 5 classi: due classi servono per contenere i dati da memorizzare, una per convertire i dati in modo da poterli fisicamente registrare nel db, una per gestire la connessione al db e una per eseguire tutte le operazioni sulle altre classi. Questa divisione permette di semplificare e specificare il lavoro di ogni elemento. I paragrafi successivi illustrano i dettagli delle classi.

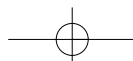
IL CODICE ALLEGATO

Gli esempi nell'articolo sono in Visual Basic; per chi fosse interessato, nel CD allegato è presente tutto il codice sorgente anche in C#.

DATI

Le due classi che contengono i dati sono veramente semplici e sono formate da una serie di campi significativi con relativi metodi get e set per inserire e recuperare le informazioni. Nell'esempio si andrà a gestire un magazzino di frutta contente vari tipi di frutta e i venditori dai quali poterla comprare. Le due classi contengono informazioni relative a questi elementi.

```
// Record
private String id;
private String name;
private String category;
private String seller;
private int sellerRecord;
private float sellerPrice;

public void setId(String data) {
    id = data;
}
...
```

Come si può osservare dal codice, è una classe

senza troppa logica. Ogni elemento è costituito da un proprio identificativo, un nome e altre informazioni relative al venditore. Anche la classe dei venditori, *Seller*, è assai simile come struttura.

```
// Seller
private String name;
private String address;
private String city;
private String state;
private String zipcode;
private String bizPhoneNumber;
private String repPhoneNumber;
private String vendor;

public void setName(String data) {
    name = data;
}
```

Ogni venditore ha un nome, un indirizzo ecc. Il codice, in questo caso, serve per mostrare l'elenco di tutti i campi che verranno inseriti nel db.
Una volta visti i dati che verrano maneggiati, si può passare agli aspetti più caratteristici del DBBerkelery iniziando con la classe che serve a convertire le due classi sopra, affinché possano essere memorizzate nel db.

CONVERSIONE DATI

In Java ci sono molti esempi di conversione dei dati da un formato a un altro. Basti pensare ai tipi primitivi e ai loro *wrapper*, come *Integer* e *int*. Conversioni più sofisticate avvengono, ad esempio, per il passaggio di oggetti in rete, sfruttando la classe *Serializable*.
Per convertire un oggetto e renderlo compatibile per la memorizzazione fisica su db, in questo esempio, si vedrà una classe specifica. La classe *RecordBinding* estende *TupleBinding* per convertire l'oggetto. Il metodo principale è quello che serve materialmente alla conversione.

```
// RecordBinding
public void objectToEntry(Object object,
                          TupleOutput to) {

    Record record = (Record)object;

    to.writeString(record.getId());
    to.writeString(record.getName());
    to.writeString(record.getCategory());
 ...
```

A ben vedere la conversione è abbastanza semplice. Chi vuole creare un nuovo oggetto, deve solo cambiare questo metodo per poterlo memorizzare. In realtà ci sono altre possibilità per memorizzare un oggetto. Nel caso fortunato in cui siano coinvolte solo stringhe, non c'è neanche bisogno di passaggi intermedi. Per tutti gli altri casi si può usare la serializzazione oppure, come nel nostro esempio, costruire un proprio convertitore. Rivedendo il nostro codice, si può notare come il metodo faccia uso dell'oggetto *TupleOutput*, lasciando intuire che dietro questa operazione c'è comunque il lavoro di altre classi realizzate dagli sviluppatori del DBBerkeley. A vantaggio di questa scelta c'è anche il lavoro specifico che consente di ottimizzare le prestazioni.
Salendo nella gerarchia, si è visto gli elementi da memorizzare e la loro conversione per poterli registrare. Ora serve preparare il db per supportare questi elementi.

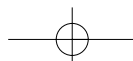
TIPI DI DATO NEL DB

La quarta classe dell'esempio ha l'obiettivo di gestire la connessione al db ed è la stessa usata in precedenza: *DBEnviroment*. In questo caso, servono altri elementi che consentono di serializzare gli oggetti che vengono memorizzati nel db. La classe *ClassCatalogDB* serve proprio a questo.

```
// DBEnviroment
classCatalogDb =
   myEnv.openDatabase(null,
      "ClassCatalogDB",
      myDbConfig);
```

Sostanzialmente, la classe non cambia di molto. Una volta visto tutti gli elementi che servono per memorizzare e gestire, si può passare alla classe che effettivamente inserisce i dati nel DB-Berkeley.

INSERIMENTO NEL DB

Per inserire i dati nel db ci sono una serie di passaggi da compiere che coinvolgono le classi viste in precedenza. Ad alto livello, le operazioni da eseguire riguardano il caricamento del db, come visto anche in precedenza. Poi si prendono i dati degli elementi da inserire nel db, leggendoli da file. Si effettua la conversione dei dati, usando la classe per la serializzazione. E, a questo punto, si può mettere tutto nel db. I dettagli del codice spiegano questi passaggi in

modo più approfondito, iniziando dal metodo *run* che riassume il flusso delle operazioni da eseguire.

```
// DBInsert
private void run(String args[])
        throws DatabaseException {

   DBEnviroment.setup(myDbEnvPath, false);
   loadSellerDb();
   loadRecordDb();
```

Come già visto in precedenza, si possono impostare molti parametri per collegarsi al db. E, alla fine, l'istruzione che permette di farlo è quella che chiama la classe *DBEnviroment*. Gli altri due metodi, *loadSellerDB* e *loadRecordDB* sono simili e servono per caricare i dati da file, convertirli e inserirli nel db.

```
// DBInsert
private void loadSellerDb()
            throws DatabaseException {

        // caricamento del file
        List sellerList = loadFile(sellerFile, 8);

        // creazione per la serializzazione
        EntryBinding dataBinding =
            new
         SerialBinding(DBEnviroment.getClassCatalog(),
                                             Seller.class);

        for (int i = 0; i < sellerList.size(); i++) {
            String[] sArray = (String[])sellerList.get(i);
            Seller seller = new Seller();
            seller.setVendorName(sArray[0]);
            seller.setAddress(sArray[1]);
            ...

            String vendorName =
                                seller.getVendorName();
            try {
                theKey = new
        DatabaseEntry(vendorName.getBytes("UTF-8"));
            } catch (IOException willNeverOccur) {}

            // serializzazione
            dataBinding.objectToEntry(seller, theData);

            // inserimento nel db
            DBEnviroment.getVendorDB().put(null,
                                        theKey, theData);
        }
    }
```

Questo metodo è il cuore della classe. Per prima cosa, con *loadFile*, si prendono i dati dei venditori da file. Poi si istanzia l'oggetto per la serializzazione. A questo punto viene scandita la lista dei venditori che vengono memorizzati negli oggetti di tipo *Seller*. Poiché il db ha bisogno di coppie chiave-valore per memorizzare i dati, si usa il nome del venditore come chiave e tutto il resto dell'oggetto come valore. Una volta creato l'oggetto, lo si serializza: ora è pronto per essere memorizzato nel db. *DBEnviroment.getVendorDB().put(theKey*, theData) serve per memorizzare il dato nel db, specificando la chiave del record. Come si può vedere, le istruzioni che effettivamente sono necessarie per inserire i dati sono poche e semplici.

Non serve a molto inserire i dati se poi non è possibile leggerli. Il prossimo esempio serve per recuperare le informazioni dal db.

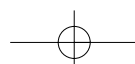
```
TriCounty Produce#309 S. Main Street#Middle Town#MN#55432#763 555 5761#Mort Dufresne#763 555 5765
Simply Fresh#15612 Bogart Lane#Harrigan#WI#53704#420 333 3912#Cheryl Swedberg#420 333 3952
Off the Vine#133 American Ct.#Centennial#IA#52002#563 121 3800#Bob King#563 121 3800 x54
The Pantry#1206 N. Creek Way#Middle Town#MN#55432#763 555 3391#Sully Beckstrom#763 555 3391
Mom's Kitchen#53 Yerman Ct.#Middle Town#MN#55432#763 554 9200#Maggie Kultgen#763 554 9200 x12
The Baking Pan#1415 53rd Ave.#Dutchin#MN#56304#320 442 2277#Mike Roan#320 442 6879
```

Fig. 3: Dati memorizzati su file

LETTURA DATI

Poiché la lettura è l'operazione speculare della scrittura, ci si può aspettare di trovare molti concetti già usati nell'esempio precedente. Infatti, c'è sempre la connessione al db. Quella che prima era la serializzazione dell'oggetto, adesso diventa deserializzazione, ma il concetto è lo stesso. L'unica differenza è il passaggio dalla scrittura alla lettura. Per chi ha dimestichezza con *JDBC*, l'operazione di lettura è analoga a quella che usa l'oggetto *ResultSet*. Per chi, invece, ama i linguaggi di programmazione dei db, la lettura sfrutta i cursori. Infine, per chi ama i dettagli, di seguito c'è il codice.

```
// DBSelect
private void run(String args[])
      throws DatabaseException {

   DBEnviroment.setup(myDbEnvPath, true);

   // serializzazione
   recordBinding = new RecordBinding();
   sellerBinding =
         new
         SerialBinding(DBEnviroment.getClassCatalog(),
            Seller.class);
   ...
   showAllRecord();
```

Proprio per cercare di mantenere la stessa

struttura dell'esempio precedente, il metodo che fa eseguire tutte le operazioni principali è, anche in questo caso, run. Le operazioni di caricamento del db e di serializzazione sono analoghe. Il metodo *showAllRecord* ha il compito di prendere i dati dal db e poi di visualizzarli.

```
// DBSelect
private void showAllRecord()
    throws DatabaseException {
  // Instanziazione del cursore
  Cursor cursor =
 DBEnviroment.getRecordDB().openCursor(null, null);

  // DatabaseEntry per recuperare i dati
  DatabaseEntry foundKey = new DatabaseEntry();
  DatabaseEntry foundData = new DatabaseEntry();

  try {
    while (cursor.getNext(foundKey, foundData,
          LockMode.DEFAULT) ==
                    OperationStatus.SUCCESS) {
      Record record =
          (Record)
            recordBinding.entryToObject(foundData);
            displayRecord(foundKey, record);
    }
  } catch (Exception e) {
  ...
```

Questo è il metodo che ha concetti nuovi rispetto a quanto visto. Da subito viene istanziato un cursore, l'oggetto che si occuperà di scandire e recuperare le informazioni dal db. Il cursore è legato al db dal quale prendere i valori. I risultati sono poi divisi in chiave e valore, come nel caso dell'inserimento. A questo punto parte il ciclo che verifica se ci sono elementi e, in caso affermativo, prende i valori e li inserisce in oggetti appropriati. Il metodo *displayRecord* ha l'obiettivo di stampare a video i risultati. E questi sono proprio i dati inseriti nel db che erano stati presi dai file nell'esempio precedente.

Con questi esempi, si hanno a disposizione gli strumenti per gestire i db. Per chi vuole andare più in profondità e gestire anche gli indici, l'esempio successivo illustra come fare.

```
AllsfruibJGK4R:
     Allspice
     fruits
     TriCounty Produce
        Number in stock: 669
        Price per unit:  0.94
        Contact:
         309 S. Main Street
         Middle Town, MN 55432
         Business Phone: 763 555 5761
         Sales Rep: Mort Dufresne
                    763 555 5765
Allsfruilfvoeg:
     Allspice
     fruits
     Simply Fresh
        Number in stock: 244
        Price per unit:  0.94
        Contact:
         15612 Bogart Lane
         Harrigan, WI 53704
         Business Phone: 420 333 3912
         Sales Rep: Cheryl Swedberg
                    420 333 3952
Allsfruio12BOS:
     Allspice
     fruits
```

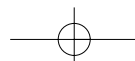
Fig. 4: Risultato della lettura dati.

INDICI

Si è visto, negli esempi precedenti, tutta la procedura per leggere e scrivere dati nel db. Le operazioni da eseguire per creare un indice sono analoghe. Questo procedimento è un po' diverso dai db normali ed è invece molto simile alle operazioni viste negli esempi precedenti. Infatti, viene creato un altro db per la gestione degli indici e poi si esegue l'operazione di creazione dell'indice su questo db che, per essere distinto dall'altro, viene chiamato db secondario. Quindi, le operazioni da eseguire per generare un indice sono due: creazione di un db secondario e creazione dell'indice.

```
// DBEnviroment
SecondaryConfig mySecConfig = new
                          SecondaryConfig();

// db secondario in modalita' read-only
mySecConfig.setReadOnly(readOnly);

// possibilita' di creare il db secondario
mySecConfig.setAllowCreate(!readOnly);
```

Come si può notare, queste istruzioni sono esattamente le stesse che si deve impostare per creare il db normale. In altre parole, il db secondario è del tutto simile a quello principale.

```
// DBEnviroment
// instanziare generatore dell'indice
ItemNameKeyCreator keyCreator =
   new ItemNameKeyCreator(new
                            RecordBinding());

// proprieta' del db secondario
```

```
mySecConfig.setSortedDuplicates(true);
mySecConfig.setAllowPopulate(true);
mySecConfig.setKeyCreator(keyCreator);

// apertura db secondario
itemNameIndexDb =
  myEnv.openSecondaryDatabase(
    null,
    "itemNameIndex",
    recordDb,
    mySecConfig);
```

Queste sono le istruzioni aggiuntive per creare l'indice. A parte le impostazioni sul db secondario, come ammettere i duplicati, l'oggetto più importante è ItemNameIndexDb, che serve materialmente per generare l'indice.
Una volta creato l'indice, servono dei metodi per sfruttarlo.

LETTURA CON INDICI

Per avere un esempio di lettura dei dati che sfrutti gli indici, si può arricchire la classe realizzata in precedenza.

```
// DBSelect
private void showItem() throws DatabaseException
                                                {

  SecondaryCursor secCursor = null;
  try {
    // definizione dell'indice
    DatabaseEntry searchKey =
      new DatabaseEntry(locateItem.getBytes
                                    ("UTF-8"));

    DatabaseEntry foundKey = new
                              DatabaseEntry();
    DatabaseEntry foundData = new
                              DatabaseEntry();

    // uso del cursore su db secondario
    secCursor =
DBEnviroment.getNameIndexDB().openSecondaryC
                              ursor(null, null);

    OperationStatus retVal =
      secCursor.getSearchKey(searchKey,
                                     foundKey,
        foundData, LockMode.DEFAULT);

    // visualizzazione risultati
    while (retVal == OperationStatus.SUCCESS) {
      Record record =
        (Record)
          recordBinding.entryToObject(foundData);
      displayRecord(foundKey, record);
      retVal = secCursor.getNextDup(searchKey,
                                     foundKey,
        foundData, LockMode.DEFAULT);
    }
  } catch (Exception e) {
  ...
```

Anche la lettura che usa gli indici è simile alla lettura normale. Viene creato un cursore, si recupera l'indice da usare, e i risultati vengono scanditi come già visto. Anche in questo caso, quindi, gli elementi aggiuntivi sono pochi.

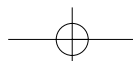
CONFIGURAZIONE

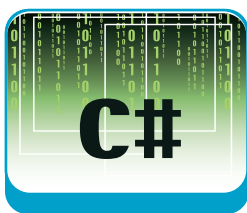
Tutti gli esempi visti necessitano di alcuni elementi per funzionare correttamente. Prima di tutto il DBBerkeley. Per installarlo, basta copiare il file .jar e mettere la variabile nel CLASSPATH. In Windows si può inserire nella pagina delle variabili d'ambiente, in Linux si può editare il file, variabile a seconda delle distribuzioni. Fatto questo, si possono lanciare i primi esempi. Il primo esempio, *VerifyDB*, serve proprio a vedere se tutto è andato bene. A seconda delle impostazioni, è necessario creare la directory *tmp* allo stesso livello della directory *classes* che contiene i file compilati. Una volta esguite queste operazioni, non resta che lanciare gli altri esempi.

CONCLUSIONI

Gli esempi visti sono un'introduzione alle capacità e alle possibilità di DBBerkeley. Si è potuto apprezzare la facilità delle operazioni di inserimento e recupero dei dati, così come la creazione di indici. Da questi esempi si capisce come DBBerkely sia un'alternativa ai db tradizionali. Sfruttando la sua caratteristica principale, di essere realizzato in Java, riesce a ottimizzare le normali operazioni di inserimento e lettura nel db. E mantiene tutte le funzionalità che un db deve avere. E può essere un ottimo strumento sia per chi, a digiuno di database e amante di Java, ha comunque la necessità di memorizzare dati, sia per chi cerca performance estreme. Non possiamo mancare in ultima analisi di fare qualche nota sull'estrema leggerezza del DB e sulla semplicità della sintassi.
Con uno sforzo realmente minimo si possono ottenere prestazioni di tutto rispetto senza allontanarci dal nostro linguaggio preferito.

Cristiano Bellucci

062-067 2-07-2007 17:57 Pagina 62

LINQ IL LINGUAGGIO SQL DENTRO .NET

IL SISTEMA DI INTERROGAZIONE DEI DATI RAPPRESENTA FORSE UNO DEI COMPONENTI PIÙ IMPORTANTI INTRODOTTI DAL .NET FRAMEWORK 3.5, IMPARIAMO AD UTILIZZARLO AFFRONTANDO ANCHE LE ALTRE NOVITÀ DEL NUOVO FRAMEWORK.

Una delle maggiori sfide dell'industria del software negli ultimi dieci anni è stata quella di raggiungere un punto di stabilità nel campo dello sviluppo software object oriented. Oggi questa sfida può definirsi vinta, grazie alla diffusione massiccia che i linguaggi object oriented hanno avuto tra gli sviluppatori. Considerato che proprio di sfide si nutre il progresso tecnologico, ed in particolar modo il mondo informatico, raggiunto il primo obiettivo ci si è posti una seconda sfida, quella di riuscire a portare i concetti della programmazione ad oggetti anche nell'ambito dell'accesso ai dati, pilastro fondamentale dello sviluppo software.

Come sappiamo, i dati che trattano le nostre applicazioni non derivano nativamente da tecnologie object oriented, spesso per la loro stessa natura come dimostrano ad esempio le informazioni contenute in database relazionali oppure le informazioni di natura XML. La gestione dei dati nelle applicazioni odierne è sempre stato un aspetto critico in quanto come ben sappiamo i dati possono provenire da diverse fonti e di conseguenza i metodi e le tecniche per accedervi sono diverse. I dati possono provenire, come abbiamo detto, da database relazionali, da file XML, ma anche da oggetti presenti in memoria (array, collection), da file Office come Excel o Word, e così via. Per ciascuno di questi domini vi è uno specifico metodo di accesso: SQL per i database, XQuery per XML, specifiche API per i documenti Office, ecc.

È a questo punto che entra in scena LINQ che appunto può essere definito come un set di *facility* che permettono, attraverso un approccio generico, di accedere in modalità object oriented a numerose sorgenti di dati che per loro natura non sono "orientate agli oggetti", ponendo al centro dell'attenzione le "operazioni" da compiere su di esse piuttosto che il modo di compierle, fornendoci un modello uniforme di accesso ai dati.

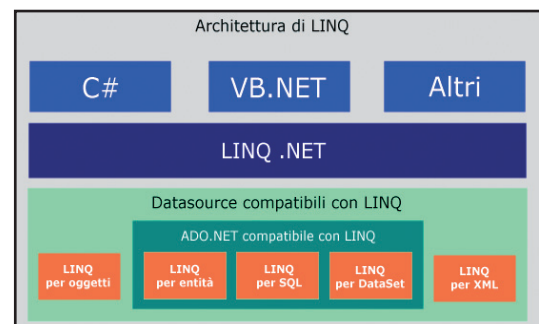



***Fig. 1:** Architettura di LINQ*

LINQ sta per **.NET Language Integrated Query** dove per *language integrated* si intende il fatto che stiamo parlando di una serie di feature aggiunte ai linguaggi di programmazione .NET esistenti (C# e VB.NET) e che permettono per l'appunto di creare delle query che possono avvantaggiarsi quindi di tutte quelle funzionalità già disponibili per i linguaggi .NET, quali ad esempio l'IntelliSense, il controllo della sintassi in fase di compilazione, e così via.

Le query LINQ possono applicarsi non solo a sorgenti di dati relazionali o XML ma anche a qualsiasi altra fonte di dati tra cui ad esempio i dati contenuti in memoria, quali Array, liste, collezioni. In generale le query LINQ possono essere applicate a qualsiasi fonte di dati di tipo *IEnumerable<T>* ovvero una qualsiasi fonte di dati enumerabile contenente collezioni di oggetti di qualunque tipo.

Un altro aspetto importante di LINQ è rappresentato dalla sua espandibilità. Infatti chiunque può espandere le funzionalità di LINQ aggiungendo nuovi operatori specifici per particolari ambiti di utilizzo oppure modificando gli operatori già esistenti per migliorarne le prestazioni o adattarne le funzionalità a campi applicativi particolari. Quest'ultima caratteristica è infatti stata usata dallo stesso team di sviluppo di LINQ per produrre diverse implementazioni che si potessero adattare alle diverse tipologie di dati da trattare. Infatti, oltre all'implementazione di base, vi

062-067 29-06-2007 16:21 Pagina 63

sono anche le implementazioni di LINQ per XML (XLinq) e quella per i database relazionali (DLinq).

INSTALLAZIONE

Dopo aver eseguito il download della **May 2006 LINQ CTP**, lanciamo il file *LINQ Preview (May 2006)*.msi e seguiamo il wizard di installazione. Ad un certo punto ci verrà chiesto se aggiornare o meno il *C# Language Service*, ovvero il componente necessario affinché nell'IDE di Visual Studio si possa ottenere il supporto dell'IntelliSense sulle query LINQ. Rispondete affermativamente, altrimenti non potrete avvantaggiarvi di una delle caratteristiche più importanti delle query LINQ ma tenete in considerazione che essendo questa una versione *Preview*, questa integrazione non è supportata da Microsoft e potrebbe in alcuni casi causare problemi con Visual Studio. Al termine dell'installazione potrete avviare Visual Studio, aprite il menu **File** e selezionate New Project... vedrete che tra i project template di Visual Basic e Visual C# ci saranno due nuove voci rispettivamente *LINQ* e *LINQ Preview* nelle quali ci sono i nuovi project template per creare applicazioni LINQ.

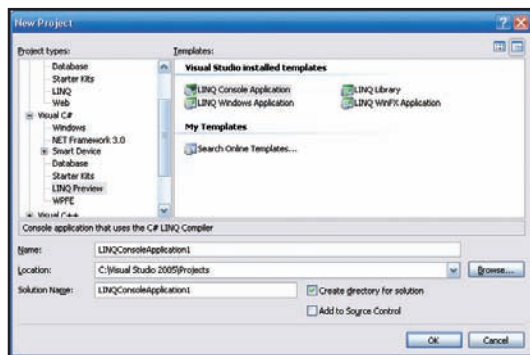



Fig. 2: Template Visual Studio per LINQ

LE NUOVE FUNZIONALITÀ DEI LINGUAGGI DEL .NET FRAMEWORK 3.5

Prima di affrontare direttamente la sintassi delle query LINQ ed il loro utilizzo è necessario fare un breve excursus tra alcune novità della versione 3.0 di C# introdotte dal .NET Framework 3.5 che è necessario conoscere per poter utilizzare LINQ visto che su queste si basa la sintassi di creazione delle query. Considerato che sia LINQ che il .NET Framework 3.5 sono ancora in fase di beta e quindi suscettibili di modifiche, è necessario tener conto del fatto che tutto quello che vedremo è valido considerando l'ultima versione disponibile di questi prodotti al momento in cui scriviamo. Ma vediamo ora una per volta le maggiori novità del .NET Framework 3.5.

LAMBDA EXPRESSIONS

Simili ai delegate e considerate una evoluzione dei metodi anonimi di C# 2.0, le Lambda Expression permettono fondamentalmente di passare del codice come parametro di un metodo.

Una lambda expression si compone di tre parti:

- un certo numero di parametri racchiusi in una parentesi tonda (la parentesi può essere omessa nel caso di un solo parametro)
- il nuovo operatore => (uguale maggiore)
- una espressione, una istruzione o genericamente un blocco di codice che sulla base dei parametri forniti, li elabora e restituisce un risultato

Vediamo quindi un esempio di Lambda Expression:

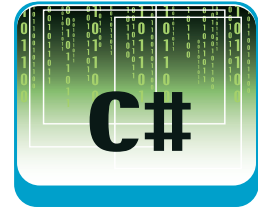
```
(int x) => x + 1
```

oppure

```
x => { return x * 2; }
```

in quest'ultimo caso possiamo omettere la parentesi perché x è l'unico parametro ed inoltre possiamo anche omettere il tipo *int* in quanto può essere implicitamente dedotto grazie alla *type inference*. Nel primo esempio la Lambda Expression è costituita da una espressione (x+1) mentre nel secondo caso è costituita da un blocco di codice ({ return x * 2; }).

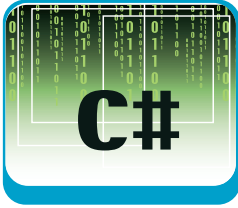
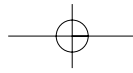
EXTENSION METHODS

Altro importante pezzo dell'architettura delle query LINQ, gli Extension Method permettono allo sviluppatore di aggiungere nuovi metodi statici pubblici ad una classe oltre a quelli già pre-

NOTA

REFERENZIARE DLL NEI PROGETTI

In Visual Studio per referenziare un assembly in un progetto è sufficiente cliccare con il tasto destro del mouse sul nome del progetto, scegliere la voce Add Reference... , cliccare sul tab Browse e selezionare dal disco rigido la DLL che s'intende referenziare nel progetto.

LE NUOVE VERSIONI DEI LINGUAGGI E I PROBLEMI DI GIOVENTÙ

LINQ si basa sulle nuove versioni dei due linguaggi di sviluppo principali del .NET Framework, ovvero C# e VB.NET. Installando LINQ, infatti, il nostro ambiente di sviluppo (Visual Studio 2005 o Visual Studio Express) sarà aggiornato con la nuova versione di C#, la 3.0 e con la nuova versione di VB.NET, la 9.0. Questi aggiornamenti portano con se diverse novità ma allo stesso tempo essendo ancora per il momento versioni in fase di sviluppo possono portare alcuni problemi di gioventù. E' quindi raccomandabile installare LINQ su una macchina di test.

senti in essa. In C# un extension method si riconosce grazie alla presenza di *this* inserito come primo parametro all'interno di una sua chiamata. Immaginiamo ad esempio di voler creare un nuovo metodo statico per la classe string che data una stringa in input ci restituisca la stessa stringa con il primo carattere in maiuscolo. Creiamo quindi la seguente classe contenente il nostro extension method:

```
namespace Test
{
        public static class StringExtensions
        {
                public static string Capitalize(this
                                        string identifier)
                                                        {
                                string newStr = "";

                        if( identifier != "")
                        {
                                newStr =
                        identifier.Substring(0,1).ToUpper() +
                                identifier.Substring(1);
                        }
                        return newStr;
                }
        }
}
```

A questo punto possiamo utilizzare il metodo *Capitalize* su una qualsiasi istanza della classe *string*:

```
string nome = "gianni";
nome = nome.Capitalize();
Console.WriteLine( nome);
```

Verrà visualizzato: Gianni

OBJECT INITIALIZATION EXPRESSIONS

È una nuova modalità per valorizzare le proprietà di una classe. Vediamo con un semplice esempio in cosa consiste. Prendiamo la classe Utente così definita:

```
public class Utente
{
  string _nome;
  string _cognome;
  bool _autorizzato = false;
  public string Nome
  {
    get { return _nome; }
    set { _nome = value; }
  }
  public string Cognome
  {
    get { return _cognome; }
    set { _cognome = value; }
  }

  public bool Autorizzato
  {
    get { return _autorizzato; }
    set { _autorizzato = value; }
  }
}
```

Quando andremo ad istanziarla, scriveremo:

```
Utente ut = new Utente();
ut.Nome = "nome utente";
ut.Cognome = "cognome utente";
ut.Autorizzato = true;
```

Con la nuova sintassi introdotta dalle *Object Initialization Expressions*, ora per istanziare la classe *Utente* e valorizzarne le sue proprietà, potremo scrivere molto più semplicemente:

```
Utente ut = new Utente {Nome = "nome utente",
  Cognome = "cognome utente", Autorizzato = true};
```

Ma l'utilità di questa nuova funzionalità è ancora più evidente nel caso delle inizializzazioni di array di oggetti. Infatti un array di tipi *Utente* potrà essere agevolmente creato in questo modo:

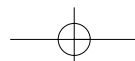
```
Utente[] utenti = {
  new Utente { Nome = "nome utente 1", Cognome
          = "cognome utente 1", Autorizzato = true },
  new Utente { Nome = "nome utente 2", Cognome
          = "cognome utente 2", Autorizzato = false },
  new Utente { Nome = "nome utente 3", Cognome
          = "cognome utente 3", Autorizzato = true },
  new Utente { Nome = "nome utente 4", Cognome
          = "cognome utente 4", Autorizzato = true },
  new Utente { Nome = "nome utente 5", Cognome
          = "cognome utente 5", Autorizzato = false },
};
```

NOTA

WEB REFERENCE

Informazioni ufficiali, documentazione ed aggiornamenti su LINQ sono disponibili sul sito:
http://msdn.microsoft.com/netframework/future/linq
Mentre all'indirizzo:
http://msdn2.microsoft.com/en-us/netframework/aa904594.aspx
trovate il sito del progetto LINQ con ulteriori informazioni sullo stesso.

ATTENTI AD ORCAS

Per poter utilizzare LINQ è necessario installare il pacchetto May 2006 LINQ CTP. Sebbene sia già disponibile una versione più aggiornata di LINQ, ovvero la March 2007 CTP, questa viene però distribuita esclusivamente con la versione full del nuovo Visual Studio Codename "Orcas" il cui pacchetto d'installazione raggiunge la ragguardevole dimensione di 4GB. È quindi preferibile eseguire il download del solo LINQ nel caso in cui non si voglia installare l'intero Visual Studio Orcas.

ANONYMOUS TYPES

I tipi anonimi sono tipi del tutto identici agli altri tipi ma con una semplice caratteristica che li rendono speciali, ovvero il fatto di non avere un "nome" che li identifichi. Vediamo con un esempio cosa significa questo.
Una dichiarazione normale di un tipo è la seguente (utilizziamo le succitate *Object Initialization Expressions*):

```
Utente ut = new Utente {Nome = "nome" Cognome
                = "cognome", Autorizzato = true};
```

Un tipo anonimo, invece viene creato nel seguente modo:

```
object ut = new {Nome = "nome" Cognome =
                "cognome", Autorizzato = true};
```

I due oggetti *ut* sono sostanzialmente identici, l'unica differenza tra loro sta nel fatto che nel secondo caso abbiamo istanziato un oggetto senza specificare un nome per esso (notare la mancanza del nome della classe dopo la keyword *new*). In entrambi i casi, però, l'istanza ut avrà tre proprietà valorizzate con i valori "nome", "cognome" e "true". A cosa serve tutto ciò? Serve in tutti quei casi in cui a noi interessano solo i valori di una certa classe (ad esempio nel caso di classi bag) e quindi con questa sintassi possiamo evitarci di creare una classe al solo scopo di utilizzarla come "contenitore".

VAR KEYWORD

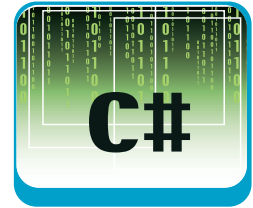
La nuova parola chiave *var* si inserisce nel discorso precedente relativo ai metodi anonimi, in quanto questa parola chiave permette per l'appunto di creare delle variabili senza specificarne il tipo. Ecco un esempio:

```
var i = 0;
var str = "Nome";
var ok = (A == B);
var num = num1 / num2;
```

In tutti questi casi i tipi delle variabili dichiarate vengono dedotti dall'espressione di inizializzazione, quindi nel caso della variabile *i* il tipo sarà *int* visto che questa è stata inizializzata con *0*. La variabile *str* invece diventerà di tipo *String* in quanto è stata inizializzata con la stringa "Nome" il cui tipo è appunto *String* e così per le altre dichiarazioni. Notiamo subito che questa keyword permette quindi di creare istanze di oggetti derivanti da tipi anonimi, infatti il precedente esempio di tipo anonimo:

```
object ut = new {Nome = "nome" Cognome =
                "cognome", Autorizzato = true};
```

diventa più correttamente:

```
var ut = new {Nome = "nome" Cognome =
                "cognome", Autorizzato = true};
```

LE QUERY LINQ

Il modo più semplice per comprendere cosa sia una query LINQ e come utilizzarla è quello di confrontarla con le query SQL. Innanzi tutto diciamo che per utilizzare gli operatori standard delle query LINQ è necessario, dopo aver installato la **May 2006 LINQ CTP**, referenziare la DLL **System.Query.dll** nella quale è presente il namespace **System.Query** che fornisce gli operatori di cui abbiamo bisogno per le query sugli oggetti. Mentre per le query su XML e su Database le DLL da referenziare saranno rispettivamente **System.Xml.XLinq.dll** e **System.Data.DLinq.dll**.

Vediamo quindi un esempio di query SQL e la sua controparte LINQ:

```
SELECT NomeProdotto FROM Prodotti WHERE Prezzo
                > 100 AND Quantita < 1000
```

Questa query recupera tutti i nomi di quei prodotti il cui prezzo è superiore a 100 e la cui quantità è inferiore alle 1000 unità. Proviamo ora a tramutare questa query in una interrogazione LINQ:

NOTA

DOVE TROVARE LINQ

Il pacchetto d'installazione May 2006 LINQ CTP ovvero il file LINQ Preview (May 2006).msi è disponibile a questo indirizzo: http://www.microsoft.com/downloads/details.aspx?familyid=1e902c21-340c-4d13-9f04 70eb5e3dceea&displaylang=en

RIABILITAZIONE DEGLI SMART TAGS

La versione May 2006 LINQ CTPè afflitta da un piccolo bug che causa, dopo la sua installazione, l'interruzione del funzionamento degli Smart Tags nell'IDE C# di Visual Studio. Per risolvere questo problema:

1. **Avviare RegEdit dal menu Esegui... di Windows**
2. **Posizionarsi sulla chiave di registro: HKEY_LOCAL_MACHINE\Software\Microsoft\VisualStudio\8.0\Packages\{A066E284-DCAB-11D2-B551-00C04F68D4DB}\SatelliteDLL**
3. **Fare doppio click con il mouse sul valore "Path" e modificarlo da "C:\Program Files\Microsoft Visual Studio 8\VC#\VCSPackages\1033\" a "C:\Program Files\Microsoft Visual Studio 8\VC#\VCSPackages\" (eliminare in pratica il 1033\)**
4. **Avviare il Prompt di Visual Studio ed eseguire il comando: devenv /setup /resetuserdata /resetsettings (questo comando resetterà tutte le impostazioni utente dell'IDE di Visual Studio)**
5. **Attendere la fine dell'esecuzione del comando e riavviare Visual Studio, a questo punto gli Smart Tags dovrebbero essere tornati a funzionare**

```
var risultato =        from p in Prodotti
                       where p.Prezzo > 100 &&
                                 p.Quantita < 1000
                       select p.NomeProdotto;
```

Come possiamo vedere, le due query sono abbastanza simili tra loro sebbene la tecnologia che vi è alla base è totalmente differente. Ed inoltre, cosa importantissima, nel caso della query LINQ avremo il supporto di Visual Studio sotto forma di Auto completamento del codice e verifica della sintassi a compile-time, cioé in fase di compilazione, cosa che non è possibile con le classiche query SQL in quanto queste nell'editor di Visual Studio sono considerate come semplici stringhe. Definita allora la query, possiamo recuperare i dati che questa fornisce attraverso un semplice ciclo *foreach*:

```
foreach( string nomeProdotto in risultato)
{
        Console.WriteLine( nomeProdotto);
}
```

od anche assegnarla come DataSource ad una GridView:

```
GridView1.DataSource = risultato;
GridView1.DataBind();
```

Il ciclo o la GridView visualizzeranno in output l'elenco dei nomi dei prodotti che soddisfano la query.
Siamo partiti dal confronto della query LINQ con una query SQL e quindi di conseguenza abbiamo presupposto che “Prodotti” fosse una tabella di un database. In realtà “Prodotti” nella query LINQ può essere diversi tipi di oggetti e non necessariamente una tabella di un database. Prodotti potrebbe essere un array di oggetti:

```
Prodotto[] Prodotti;
```

una collection generica:

```
List<Prodotto> Prodotti = new List<Prodotto>();
```

una DataTable:

```
DataSet dsProdotti = GetProdottiDataSet();
DataTable Prodotti = dsProdotti.Tables[“Prodotti”];
```

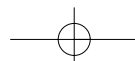
oppure un nodo Xml:

```
XElement.Load(Server.MapPath("/") +
                    "Prodotti.xml").Elements("Prodotto")
```

A seconda poi del tipo di sorgente dati sulla quale andremo ad eseguire la query potremo utilizzare degli operatori specifici per quella particolare sorgente e quindi potremo utilizzare il namespace System.Xml.XLinq se stiamo trattando dati XML oppure il namespace System.Data. DLinq se stiamo trattando tabelle di Database. Un altro aspetto importante delle query LINQ è il momento in cui queste vengono realmente eseguite. Una query LINQ, infatti, non viene eseguita se non quando non vengono richiesti i risultati della stessa. Quindi la query viene realmente eseguita solo allorquando viene richiamato il codice (ad esempio il *foreach*) che accede ai suoi risultati.
Ecco ora un esempio di query su oggetti in memoria, nello specifico su collection generiche:

```
var clienti = new List<Cliente>();

clienti.Add(new Cliente("Gianni", "Bari", "Corso Italia,
                                          10", "BA"));
clienti.Add(new Cliente("Mario", "Milano", "Piazza
                                    Dante, 123", "MI"));
clienti.Add(new Cliente("Clara", "Bari", "Via Vittorio
                                  Veneto, 45/c", "BA"));
clienti.Add(new Cliente("Marco", "Bologna", "Via
                              Marsala, 120/a", "BO"));
var clientiBari = from c in clienti
                  where c.Provincia == "BA"
                  select c;
 Console.WriteLine("Clienti della provincia di Bari:");

foreach (var c in clientiBari)
{
        Console.WriteLine(c.Nome);
}
```

dove la classe Cliente è la seguente:

```
public
 {
   private _nome = "";
   private _citta = "";
   private _indirizzo = "";
   private _provincia = "";

   public Cliente( nome, citta, indirizzo, provincia)
   {
    _nome = nome;
    _citta = citta;
    _indirizzo = indirizzo;
    _provincia = provincia;
   }

   public Nome
    { get { return _nome; } set{ _nome = value;}}
```

```
  public Provincia
   { get { return _provincia; } set{ _provincia =
                                          value;} }
 }
```

In questo caso abbiamo creato una *List* generica di tipi *Cliente* e poi attraverso la query clientiBari abbiamo estrapolato tutti gli oggetti *Cliente* la cui proprietà *Provincia* è uguale a "Bari".
Naturalmente *where* è solo uno dei tanti operatori possibili messi a disposizione da LINQ, infatti tra gli altri troviamo anche l'operatore *orderby*, utilizzato per ordinare i dati e l'operatore *groupby* utilizzato per raggrupparli. Ecco un esempio di *orderby*:

```
var risultato = from c in clienti
                   orderby c.Provincia
                   select c;
```

ed uno di groupby:

```
var risultato = from c in clienti
group c by c.Provincia into g
select new { Provincia = g.Key, Totale = g.Count() };
```

in quest'ultimo esempio viene restituita una collezione di oggetti (nello specifico si tratta di un Anonymous Type) aventi due proprietà, la prima *Provincia* valorizzata con la provincia per cui si è raggruppato ed una seconda proprietà, *Totale*, nella quale avremo il totale dei clienti in quella provincia.

OPERATORI STANDARD DELLE QUERY LINQ

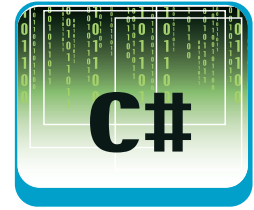
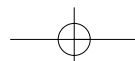
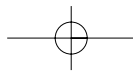
Ecco un elenco sintetico di alcuni tra i principali operatori standard di LINQ che è possibile utilizzare da subito appena installato **Orcas** o la **May 2006 LINQ CTP**:

- **Where**
  Operatore utilizzato per filtrare i dati
- **Select/SelectMany**
  Operatore per la selezione dei dati
- **Join/GroupJoin**
  Operatore utilizzato per effettuare join
- **Concat**
  Operatore di concatenazione
- **OrderBy/ThenBy/ OrderByDescending/ ThenByDescending**
  Operatori di ordinamento
- **Reverse**
  Operatore utilizzato per invertire l'ordinamento
- **GroupBy**
  Operatore di raggruppamento
- **Distinct**
  Operatore utilizzato per rimuovere i dati duplicati
- **ToArray/ToList**
  Operatore di conversione utilizzato per restituire un array o una List<T>
- **First/FirstOrDefault/Last/LastOrDefault/ Single/SingleOrDefault**
  Operatori utilizzati per fornire solo alcuni elementi particolari del risultato della query (il primo, l'ultimo, ecc)
- **ElementAt/ElementAtOrDefault**
  Operatori utilizzati per restituire solo gli elementi in una certa posizione nel risultato
- **DefaultIfEmpty**
  Operatore che restituisce un particolare valore nel caso un certo campo del risultato sia nullo
- **Range**
  Restituisce un numero in un range di valori
- **Repeat**
  Ritorna un certo numero di valori prestabiliti
- **Empty**
  Ritorna una sequenza di valori nulli
- **Contains**
  Verifica la presenza di un elemento in un gruppo di valori
- **Count/LongCount**
  Operatori di aggregazione che forniscono il numero di elementi restituiti dalla query
- **Sum/Min/Max/Average**
  Operatori di aggregazione che restituiscono la somma, il minore, il maggiore o la media tra gli elementi restituiti dalla query

CONCLUSIONI

Non tutti i purisi di .NET hanno da subito apprezzato tutte le novità introdotte dal nuovo framework, tuttavia ad un'analisi approfondita risulta evidente che LINQ è uno strumento davvero potente che ci permetterà di sviluppare applicazioni molto più solide dal punto di vista dell'accesso ai dati e sicuramente molto più efficienti.
Naturalmente si dovrà fare un po' di pratica con le novità portate dalla nuova versione di C# e VB.NET ma dopo questo breve periodo di apprendimento ci renderemo conto di utilizzare una tecnologia che ci faciliterà di molto il lavoro. Tutto sta ora ad attendere la versione definitiva di "Orcas" per poter finalmente iniziare a sviluppare con strumenti completi e stabili.

Gianni Malaga

L'AUTORE

Gianni Malanga
Lavora da più di 8 anni con tecnologie Microsoft in particolare nel campo dello sviluppo Web. Sviluppa in .NET ormai da alcuni anni e ha svolto diverse docenze per corsi di formazione professionali. Dopo aver lavorato alle dipendenze di importanti realtà locali, attualmente ha intrapreso la libera professione costituendo la ditta individuale Kaone Consulting che si occupa di consulenza informatica per PMI su tutto il territorio nazionale.
Gli si può scrivere all'indirizzo kaone@email.it **e il suo blog è disponibile su:** http://thekaone.blogspot.

UN SISTEMA DI LOG SOFISTICATO IN VB6

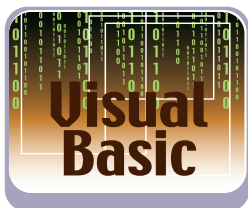
FORNIRE UN SISTEMA DI LOG BASATO SU FILE ALLE APPLICAZIONI È COSA BANALE, MA RILEGGERE, CLASSIFICARE E GESTIRE LE INFORMAZIONI TRACCIATE È TUTT'ALTRO CHE SEMPLICE VEDIAMO COME SCRIVERLE E RILEGGERLE DA DB, XML E FILE DI TESTO

REQUISITI

Conoscenze richieste
Visual Basic 6 livello intermedio

Software
Microsoft Windows 2000 o successivi e Microsoft Visual Basic 6.

Impegno

Tempo di realizzazione

In questo articolo ci occuperemo della realizzazione di una mini-architettura di Log che sfrutti al meglio la vecchia tecnologia Visual Basic per realizzare un sistema potente, flessibile e poliedrico per il tracciamento di log applicativi. L'idea è quella di realizzare un sistema di log independente dal device fisico sul quale si andrà a tracciare l'informazione, attraverso un sistema di polimorfismo per interfaccia, ma che abbia una potente caratteristica aggiuntiva: la possibilità di effettuare ricerche granulari ed efficaci sul log per una successiva rilettura delle informazioni tracciate, sempre però disinteressandosi del formato fisico di salvataggio del log. In pratica se si scrive in un log senza sapere dove vanno a finire queste informazioni perché il sistema può agganciare dinamicamente un driver per un differente device fisico di log (database, xml o testo puro) e si può successivamente rileggere queste informazioni o effettuare ricerche puntuali, senza avere direttamente a che fare con il formato fisico di salvataggio…
Il codice è scritto in Visual Basic 6 sfruttando diverse tecnologie per differenti device fisici (ADO 2.5 per i database, MSXML 3.0 per il formato xml dei log ed infine il motore VB Script Regular Expression per effettuare ricerche nel device di log basato su file di testo). Troppa grazia? No… State a guardare.

LA CLASSE LOGMANAGER

Il cuore del nostro sistema è l'oggetto LogManager, dell'interfaccia ILogStorage.
Questa interfaccia deve essere implementata ogni volta che si vuole introdurre un nuovo device di log. Per cui avremo, nell'esempio specifico, tre librerie DLL COM rispettivamente per il log su database, per il log su XML e per il log su file di testo.

L'oggetto LogItem rappresenta la voce atomica di log che si intende salvare; esso contiene tutte le tipiche informazioni di un sistema di log quali data, ora, codice di errore o di segnalazione, messaggio testuale, mittente, applicazione che ha originato il messaggio, ecc... Ogni voce è identificata da una chiave univoca basata su un GUID in modo che possa essere successivamente identificabile e recuperabile.
Ma procediamo finalmente con un esempio di scrittura del log:

```
Dim olog As Amarcord.LogManager
Dim oLogItem As Amarcord.logItem

Set olog = New Amarcord.LogManager
If Not olog.Init Then
  End
End If
Set oLogItem = New Amarcord.logItem
With oLogItem
    .Application = "Prova per CP"
    .DebugInfo = "frmLogMain - Command1_Click()"
    .LogLevel = llWarning
    .MessageCode = 255
    .MessageText = "Questa è solo una  prova per CP"
    .Sender = "Vito Vessia"
    .SessionID = "Session 1324"
    .Properties("ciro").Value = "wew"
    .Properties("ciro1").Value = "wewsdfsd"
    .SetDate Now
End With
olog.log oLogItem
```

Il logItem è impostato al LogLevel llWarning. Accadrà quindi che il livello di log del LogManager (proprietà LogLevel) è impostato con una priorità minima più alta, la richiesta di tracciamento del log verrà semplicemente ignorata. Ad esempio, se il LogManager è impostato in modalità llError, tutte le richieste di log di livello llInfo o llWarning verranno ignorate.

068-075 29-06-2007 16:23 Pagina 69

RECUPERO DELLE INFORMAZIONI DI LOG

Una interessante caratteristica di questo sistema di log è la capacità di recuperare successivamente le informazioni scritte. Innanzitutto questo è possibile grazie alla presenza dell'Id in ogni voce di log, infatti un primo metodo del LogManager è proprio GetLog che accetta un Id di log e restituisce, se esiste, il logItem corrispondente:

```
Dim oLogItem As LogItem
Set oLogItem =
  oLogManager.GetLog("40104E77AF494010A171466D
                                   FbE946611")
```

Un altro modo per recupare le informazioni di log è quello di cercare per un range di date, operazione possibile grazie al fatto che ogni log item conserva la data e l'archiettura di log consente di cercare log item per data. Ma osserviamo il seguente esempio:

```
Dim ologs As Amarcord.LogItems
Set ologs = oLogManager.GetLogs(Now - 1, Now +
                                    2, False)
```

I primi due parametri fanno chiaramente riferimento al range di date da cercare il terzo, invece, indica se nella ricerca deve essere considerata anche la parte time della data. Questa funzionalità selettiva è interessante se si intende recuperare il log relativo ad un'intera giornata o solo da una certa ora in poi di quella giornata.

L'INTERFACCIA ILOGSTORAGE

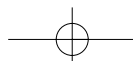
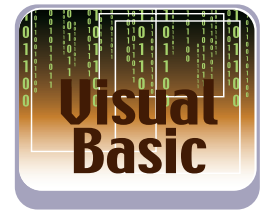
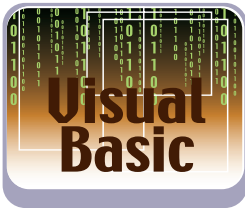
Fino ad ora abbiamo osservato come implementare un normale sistema di log con delle facility aggiuntive date dalla possibilità di recuperare una voce di log dall'Id o dalla data. Ma l'aspetto più interessante è il carattere polimorfico e flessibile degli storage di log: il sistema infatti non segna le informazioni sempre nello stesso modo, ma si avvale di un sistema di interfacce che possono donare al sistema di log nuove e diverse forme di salvataggio. Nel caso specifico implementeremo tre differenti dispositivi di log: su database (Microsoft Access), su file xml e su file di testo.
Quindi ogni metodo del LogManager redirige le richieste sull'oggetto di log ILogStorage sottostante fungendo semplicemente da passante. Osserviamo infatti l'implementazione del metodo Log di LogManager:

```
Public Function Log(LogItem As LogItem, _
```

```
            Optional ByVal LogDevice As Integer
                              = -1) As Boolean
    Dim lStorage As ILogStorage

    If m_eLogLevel < LogItem.LogLevel Then
        Exit Function
    End If
    Log = True
    If LogDevice = -1 Then
        For Each lStorage In m_StorageColl
            Log = Log And lStorage.Log(LogItem)
        Next
    Else
        Set lStorage = m_StorageColl("K" &
                                    LogDevice)
        Log = Log And lStorage.Log(LogItem)
    End If
End Function
```

Si può osservare come la richiesta di log viene semplicemente passata al log storage sottostante richiamato in modo polimorfico per interfaccia. Il LogManager non è già a conoscenza a priori dei log storage su cui agire, ma questi devono essere registrati dinamicamente dall'esterno. I log storage vengono istanziati, inizializzati e poi registrati nel sistema attraverso il metodo RegisterStorage a cui viene passato l'oggetto e la chiave con cui lo storage verrà rappresentato nel log manager. Questa chiave è importante perché dall'esterno si potrà chiedere esplicitamente di agire su un determinato storage attraverso il parametro StorageKey opzionalmente passato alla maggior parte dei metodi di LogManager. Il metodo Log di LogManager mostra proprio un esempio di questo utilizzo: se il parametro StorageKey viene passato, l'informazione di log viene scritta direttamente sul log storage corrispondente, altrimenti verrà tracciata su tutti gli storage registrati nel log manager.

IL LOG STORAGE SU DATABASE

È stato adoperato un database Access 2000 basato su una sola tabella, Logs, la cui struttura è mostrata in Figura 1. I campi della tabella sono molto prevedibilmente rimappati sulle proprietà dell'oggetto LogItem ed infatti osserviamo l'implementazione del metodo Log dell'interfaccia IlogStorage. D'altro canto anche il metodo di recupero di un log item è altrettanto semplice perché basato su una select nella tabella usando la chiave log_id = id da cercare, come si può osservare dal codice sorgente allegato. In Figura 2 è possibile osservare un semplice client Visual Basic 6 che sfrutta il sistema di log: è possibile

infatti scrivere nuove voci di log scegliendo direttamente lo storage su cui agire e recuperare delle voci di log dagli storage per id o per range di date. È interessante notare i tempi di recupero delle informazioni dal db: effettuare una ricerca su un range di date che recupera 900 log item dal db, crea le relative istanze di log item e restituisce la collection dei risulati ed infine popola la listview con queste informazioni richiede meno di 100 millisecondi. Sono prestazioni che è ovvio aspettarsi da uno storage basato su database e che quindi fa uso di select SQL per recuperare le informazioni. Sarà interessante osservare le prestazioni di recupero per gli altri due storage.

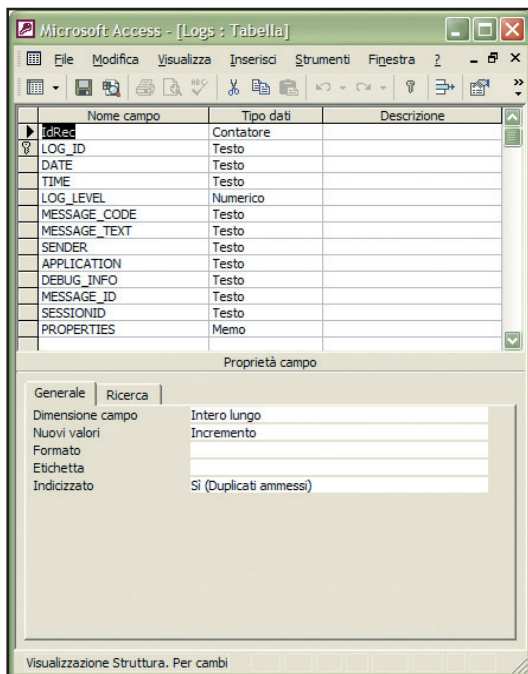



Fig. 1: La struttura della tabella di Log

Fig. 2: Il client in azione

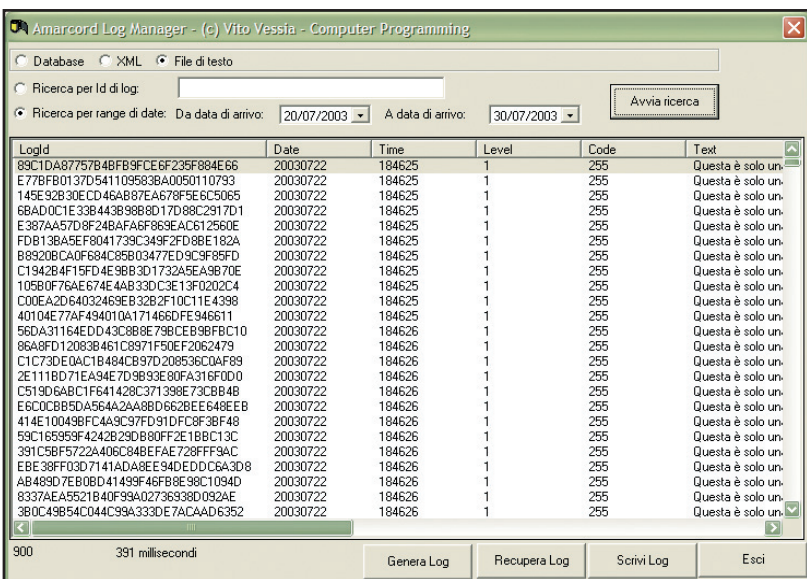
LOG SU XML

L'xml è un formato ricco, semplice e strutturato che si presta al salvataggio, alla conservazione e al recupero di informazioni complesse. Per il nostro sistema di log si è pensato di strutturare un nodo xml per ciascuna voce di log, definito come di seguito:

```
<MMCOMM-LOG>
  <LOGITEM
   LOG_ID="89C1DA87757B4BFB9FCE6F235F884E66"
  DATE="20030722" TIME="184625" LOG_LEVEL="1"

          MESSAGE_CODE="255"
 MESSAGE_TEXT="Questa &#232; solo una prova per
                                            CP"
          SENDER="Vito Vessia" APPLICATION="Prova
                                         per CP"

          DEBUG_INFO="frmLogMain -
                Command1_Click()" MESSAGE_ID=""
                           SESSIONID="dfsdfdsfdfs"
PROPERTIES="CIRO&lt;^!^&gt;wew&lt;^&amp;^&gt
              ;CIRO1&lt;^!^&gt;wewsdfsd" />
</MMCOMM-LOG>
```

L'xml è un formato testuale, ma nessuno può veramente prendere in considerazione l'idea di scrivere xml come se fosse un file di testo; ci si affida così ai DOM e l'implementazione Microsoft in tecnologia COM è di notevole rilievo.

Adotteremo la versione 3.0 che, pur non essendo l'ultima a disposizione, è certamente la più diffusa. Dunque è necessario referenziare la libreria Microsoft XML, v3.0. Osserviamo dunque l'implementazione del metodo Log dell'interfaccia ILogStorage:

```
Private Function ILogStorage_Log(LogItem As
                              Amarcord.LogItem, _
                                       Optional
 ByVal LogStorageSettings As String = "") As Boolean
    ILogStorage_Log = LogOnXML(LogItem)
End Function

Private Function LogOnXML(LogItem As LogItem, _
                   Optional XmlFileName As String
                              = "") As Boolean

    Dim l_oDoc As MSXML2.DOMDocument
    Dim l_oDocElement As MSXML2.IXMLDOMElement
    Dim l_sXMLFile As String

    On Local Error GoTo ErrorHandler
    If XmlFileName <> "" Then
        l_sXMLFile = XmlFileName
    Else
```

```
            l_sXMLFile = m_sXMLFile
        End If
        Set l_oDoc = New MSXML2.DOMDocument
        l_oDoc.Load l_sXMLFile
        LogItem.LogDate = IIf(LogItem.LogDate = "",
            Format(Now, "YYYYMMDD"), LogItem.LogDate)
        LogItem.LogTime = IIf(LogItem.LogTime = "",
            Format(Time, "HHMMSS"), LogItem.LogTime)
        With l_oDoc
            If Not l_oDoc.documentElement Is Nothing
                                                Then
                If l_oDoc.documentElement.baseName <>
                                    "MMCOMM-LOG" Then
                    Exit Function
                End If
            Else
                l_oDoc.createElement "MMCOMM-LOG"
                l_oDoc.appendChild
                    l_oDoc.createElement("MMCOMM-LOG")
            End If
            Set l_oDocElement =
                    l_oDoc.createElement("LOGITEM")
            With l_oDocElement
                .setAttribute "LOG_ID",
                        HTMLEncode(LogItem.LogID)
                .setAttribute "DATE",
                        HTMLEncode(LogItem.LogDate)
                .setAttribute "TIME",
                        HTMLEncode(LogItem.LogTime)
                .setAttribute "LOG_LEVEL",
                        HTMLEncode(LogItem.LogLevel)
                .setAttribute "MESSAGE_CODE",
                        HTMLEncode(LogItem.MessageCode)
                .setAttribute "MESSAGE_TEXT",
                        HTMLEncode(LogItem.MessageText)
                .setAttribute "SENDER",
                        HTMLEncode(LogItem.Sender)
                .setAttribute "APPLICATION",
                        HTMLEncode(LogItem.Application)
                .setAttribute "DEBUG_INFO",
                        HTMLEncode(LogItem.DebugInfo)
                .setAttribute "MESSAGE_ID",
                        HTMLEncode(LogItem.MessageID)
                .setAttribute "SESSIONID",
                        HTMLEncode(LogItem.SessionID)
                .setAttribute "PROPERTIES",
                        HTMLEncode(LogItem.GetPropertyStream)
            End With
            l_oDoc.documentElement.appendChild
                                        l_oDocElement
            l_oDoc.save l_sXMLFile
        End With
ErrorHandler:
        LogOnXML = (Err = 0)
End Function
```

In Figura 3 è possibile osservare un log xml generato dal nostro sistema.

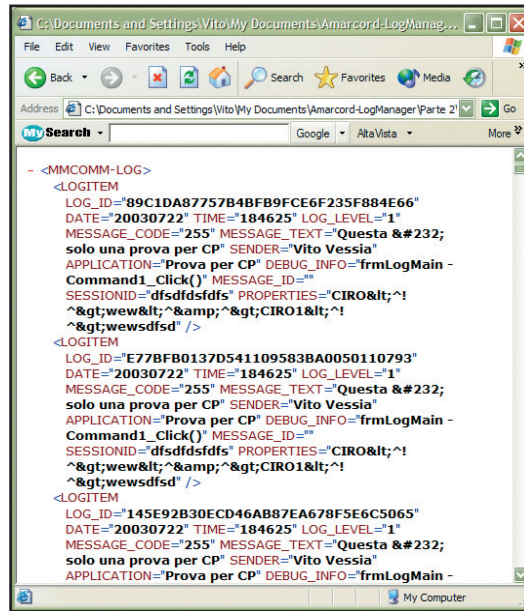
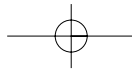
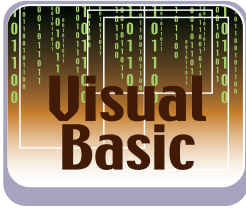



Fig. 3: Il mockup html del nostro form, editato con Nvu

RECUPERO DI INFORMAZIONI DI LOG DALLO STORAGE XML

Diventa invece decisamente più interessante comprendere come implementare i meccanismi di ricerca e recupero delle informazioni di log per id e per range di date (i metodi GetLog e GetLogs dell'interfaccia ILogStorage e quindi del LogManager). In questo ci viene incontro una potente tecnologia che correda lo standard xml: Xpath [1]. Si tratta di un linguaggio di interrogazione di dati da xml che ha poco da invidiare ad analoghi quali SQL per i database relazionali. Probabilmente, a differenza di quest'ultimo, se da una parte prevede un numero inferiore di costrutti e di funzioni di ricerca, dall'altra sfrutta una peculiarità insita nell'xml e complementamente assente, invece, nei database relazionali: la struttura gerarchica. In pratica è possibile effettuare ricerca scendendo in profondi nei sottonodi e quindi non solo in maniera flat o simulabile attraverso join come avviene per SQL. Osserviamo quindi come deve essere strutturata una query XPath che vuole individuare un logitem con id E77BFB0137D541109583BA0050110793:

```
/MMCOMM-
   LOG/LOGITEM[@LOG_ID='E77BFB0137D541109583B
                                   A0050110793']
```

È molto semplice ma potente, va letto come: parti dal nodo MMCOMM-LOG, scendi sul sottonodo LOGITEM e cerca tutti i nodi LOGITEM che hanno per attributo LOG_ID= E77BFB0137D541109583BA0050110793. Non sarà

068-075 29-06-2007 16:23 Pagina 72

una sorpresa, quindi, che in pochissimi istanti, anche su file xml di notevole dimensione, il DOM XML di Microsoft è in grado di recuperare rapidamente l'informazione che ci serve! Per effettuare l'esecuzione della query che ci restituisce un elenco di nodi, ci avvaliamo del metodo selectNodes dell'oggetto MSXML.DOMDocument:

```
Dim l_oDoc As MSXML2.DOMDocument
Dim l_oNodeList As MSXML2.IXMLDOMNodeList
Set l_oDoc = New MSXML2.DOMDocument
l_oDoc.Load l_sXMLFile
Set l_oNodeList = l_oDoc.selectNodes("/MMCOMM-
                LOG/LOGITEM[@LOG_ID='" & LogID & "']")
```

L'implementazione del metodo GetLog del driver xml permette semplicemente di deserializzare il nodo Xml restituito dalla query Xpath in un oggetto LogItem. Diventa leggermente più complessa, invece, la query XPath in grado di estrarre un gruppo di logitem per range di date. Supponiamo di voler cercare tutti i log dal 20/7/2003 al 30/7/2003 senza voler considerare la parte time della data:

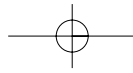
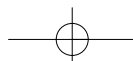
```
/MMCOMM-LOG/LOGITEM[number(@DATE) >=
           20030720 and number(@DATE) <= 20030730]
```

Questa nuova query introduce l'uso delle funzioni XPath ed in particolare della funzione number che converte una stringa in un numero; infatti la query chiede proprio di convertire l'attributo DATE in numero e di confrontarlo rispetto ad una data in formato ISO YYYYMMDD che altro non è che un numero. Inoltre vengono messe in AND due condizioni. Anche in questo caso il risultato è immediato. Se in precedenza avevamo detto che effettuare una ricerca per data nello storage di log basato su db access per tirare fuori 900 item di log ha richiesto meno di 140 millisecondi e di certo questa informazione non può avervi sconvolti abiutati come siete alle prestazioni notevoli dei database, anche client, non potrete invece restare altrettanto indifferenti alle prestazioni dell'analoga interrogazione su driver di log xml che restituisce sempre 900 log item. Sembra incredibile, ma il DOM XML e l'XPath in questa implementazione hanno raggiunto risultati degni di rilievo se confrontati con l'omologo Access. Per l'osservazione della GetLogs si rimanda al sorgente allegato.

RECUPERO DI INFORMAZIONI DI LOG DALLO STORAGE SU FILE DI TESTO

Abbiamo visto come utilizzare xml come un potente database di log in grado non solo di salvare le notifiche di eventi applicativi delle nostre applicazioni, ma anche di recuperare successivamente tali informazioni attraverso ricerche rapide e mirate. In effetti il formato xml, per la sua strutturatezza, si presta a questo tipo di utilizzo, ma certamente non si potrà dire lo stesso per la forma più antica di salvataggio di log applicativi: i file di testo. Osserviamo come appare una tipica voce di log tracciata su file attraverso il terzo driver di log:

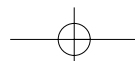
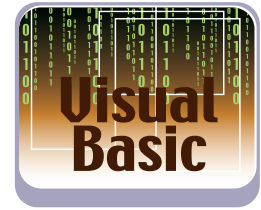
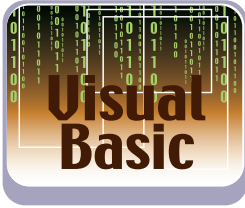
```
LOGID="E77BFB0137D541109583BA0050110793"
                LEVEL=1 DATETIME="20030722-184625"
   CODE=          255
   TEXT=Questa è solo una  prova per CP
   SENDER=        Vito Vessia
   APPLICATION=Prova per CP
   MESSAGEID=
   SESSIONID=     dfsdfdsfdfs
   PROPERTIES=
         CIRO<^!^>wew<^&^>CIRO1<^!^>wewsdfsd
   DEBUG_INFO=    frmLogMain - Command1_Click()
================================
            ===========================
```

A differenza del formato xml e del salvataggio su database, il log testuale ha il vantaggio dell'estrema semplicità e dell'essere fatto per essere già fruito as is, cioè senza alcuna rielaborazione successiva o senza alcuna applicazione di layout per la visualizzazione. È certamente più difficile infatti leggere un file xml o aprire un database Access, soprattutto perché nel secondo caso è prevista la disponibilità di Access sulla macchina. Si rende così necessario scrivere dei visualizzatori di log; con il log testuale tutto questo non è necessario: se si intende leggere un log, basta aprirlo con Notepad!
Il layout scelto nell'esempio non è particolarmente sofisticato o gradevole dal punto di vista estetico, ma sicuramente efficace a titolo esemplificativo. Osserviamo il metodo Log dello storage di log in questione che è in grado di scrivere una voce di log:

```
Private Function ILogStorage_Log(LogItem As
                                Amarcord.LogItem, _
                                     Optional
    ByVal LogStorageSettings As String = "") As Boolean
   ILogStorage_Log = LogOnText(LogItem)
End Function

Private Function LogOnText(LogItem As LogItem, _
                 Optional LogFileName As String
                                    = "") As Boolean
   Dim l_sTextFile As String
   Dim l_iFileNo As Integer
   Dim l_sLogText As String

   On Local Error GoTo ErrorHandler
   If LogFileName <> "" Then
```

```
        l_sTextFile = LogFileName
    Else
        l_sTextFile = m_sTextFile
    End If
    l_iFileNo = FreeFile
    LogItem.LogDate = IIf(LogItem.LogDate = "",
            Format(Now, "YYYYMMDD"), LogItem.LogDate)
    LogItem.LogTime = IIf(LogItem.LogTime = "",
            Format(Time, "HHMMSS"), LogItem.LogTime)
    Open l_sTextFile For Append As l_iFileNo
    l_sLogText = "LOGID=" & """" & LogItem.LogID &
            """" & " LEVEL=" & LogItem.LogLevel & _
            " DATETIME=" & """" & LogItem.LogDate
            & "-" & LogItem.LogTime & """" & vbCrLf & _
            "    CODE=" & vbTab &
                LogItem.MessageCode & vbCrLf & _
            "    TEXT=" & vbTab &
                LogItem.MessageText & vbCrLf & _
            "    SENDER=" & vbTab &
                LogItem.Sender & vbCrLf & _
            "    APPLICATION=" &
                LogItem.Application & vbCrLf & _
            "    MESSAGEID=" & vbTab &
                LogItem.MessageID & vbCrLf & _
            "    SESSIONID=" & vbTab &
                LogItem.SessionID & vbCrLf & _
            "    PROPERTIES=" & vbTab &
            LogItem.GetPropertyStream & vbCrLf & _
            "    DEBUG_INFO=" & vbTab &
                LogItem.DebugInfo & vbCrLf & _
"================================
    =============================" &
                vbCrLf & vbCrLf
    Print #l_iFileNo, l_sLogText
    Close #l_iFileNo
ErrorHandler:
    LogOnText = (Err = 0)
End Function
```

Si tratta di codice BASIC decisamente old fashioned che ci riporta indietro di 30 anni… apri un file di testo e ci scrivi dentro. Certamente più complessa è l'operazione che segue: la ricerca e il recupero di informazioni dal file di testo. Sembrerebbe all'apparenza un'operazione così complicata, da risultare quasi impossibile: bisognerebbe implementare un parser di testo di complessità non banale in grado effettuare la scansione e la ricerca di informazioni dal log.
Ma fortunatamente esiste una tecnologia già pronta all'uso: le Regular Expression. In sintesi si tratta, per chi già non la conoscesse, di un notazione sintattica, di media complessità, in grado di esprimere criteri di ricerca su testi basati sull'occorrenza di pattern, cioè di parti di testo ripetitive. Microsoft ha fornito con Internet Explorer e in tecnologia COM un motore di espressioni regolari molto potente. Si tratta della libreria Microsoft VBScript Regular Expression 5.5 fornito con Internet Explorer 6 o con la libreria Microsoft Script Engine 5.5 e successive.
Così, rimandando agli articoli e alle fonti specifiche sull'argomento, limitiamoci ad introdurre il pattern RegEx in grado di implementare la GetLog, cioè di recuperare una voce di log a partire da un certo Id di log:

```
LOGID="(40104E77AF494010A171466DFE946611)"\
        s*LEVEL=(\d{1})\s*DATETIME="(.*)-
(.*)"\r\n\s*CODE=\s*(.*)\r\n\s*TEXT=\s*(.*)\r\n\s*
SENDER=\s*(.*)\r\n\s*APPLICATION=\s*(.*)\r\n\s*
MESSAGEID=\s*(.*)\r\n\s*SESSIONID=\s*(.*)\r\n\s
*PROPERTIES=\s*(.*)\r\n\s*DEBUG_INFO=\s*(.*)
```

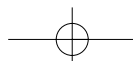
Nella struttura dell'espressione regolare si possono notare le varie parti costitutive del messaggio di log (log id, data, codice, ecc…). Ciascuna delle parti viene identificata e messa in un "gruppo" (ad esempio SENDER=\s*(.*)). Ogni gruppo è rappresentato da una sottoespressione regex racchiusa tra parentesi tonde. Il valore di ciascun gruppo estratto da una stringa è richiamabile attraverso il modello ad oggetti di RegEx. Osserviamo infatti l'implementazione del metodo GetLog:

```
Private Function ILogStorage_GetLog(ByVal LogID As
                                        String, _
Optional ByVal LogStorageSettings As String = "") As
                                  Amarcord.LogItem
    Set ILogStorage_GetLog = GetLogFromText(LogID)
End Function
Public Const REGEX_GET_LOGITEM As String = _
"LOGID=""(<LOG_ITEM>)""\s*LEVEL=(\d{1})\s*DATE
                                      TIME=""(.*)-
(.*)""\r\n\s*CODE=\s*(.*)\r\n\s*TEXT=\s*(.*)\r\n\s*S
ENDER=\s*(.*)\r\n\s*APPLICATION=\s*(.*)\r\n\s*MES
SAGEID=\s*(.*)\r\n\s*SESSIONID=\s*(.*)\r\n\s*PROP
          ERTIES=\s*(.*)\r\n\s*DEBUG_INFO=\s*(.*)"
Private Function GetLogFromText(ByVal LogID As
                                        String, _
                      Optional ByVal LogFileName As
                              String = "") As LogItem
    Dim l_oRegExp As VBScript_RegExp_55.RegExp
    Dim l_oMatchColl As
                  VBScript_RegExp_55.MatchCollection
    Dim l_oMatch As VBScript_RegExp_55.Match
    Dim l_sTextFile As String
    Dim l_sText As String
    Dim l_oTextStream As Scripting.TextStream
    Dim l_oFSO As Scripting.FileSystemObject
    On Local Error GoTo ErrorHandler
    If LogFileName <> "" Then
        l_sTextFile = LogFileName
    Else
        l_sTextFile = m_sTextFile
    End If
```

```
Set l_oFSO = New Scripting.FileSystemObject
Set l_oTextStream =
        l_oFSO.OpenTextFile(l_sTextFile, ForReading)
l_sText = l_oTextStream.ReadAll
Set l_oRegExp = New VBScript_RegExp_55.RegExp
l_oRegExp.Pattern = Replace(REGEX_GET_LOGITEM,
                        "<LOG_ITEM>", LogID)
Set l_oMatchColl = l_oRegExp.Execute(l_sText)
If l_oMatchColl.Count > 0 Then
    Set l_oMatch = l_oMatchColl(0)
    Set GetLogFromText = New LogItem
    GetLogFromText.LogID =
                        l_oMatch.SubMatches(0)
    GetLogFromText.LogDate =
                        l_oMatch.SubMatches(2)
    GetLogFromText.LogTime =
                        l_oMatch.SubMatches(3)
    GetLogFromText.LogLevel =
                        l_oMatch.SubMatches(1)
    GetLogFromText.MessageCode =
                        l_oMatch.SubMatches(4)
    GetLogFromText.MessageText =
                        l_oMatch.SubMatches(5)
    GetLogFromText.Sender =
                        l_oMatch.SubMatches(6)
    GetLogFromText.Application =
                        l_oMatch.SubMatches(7)
    GetLogFromText.DebugInfo =
                        l_oMatch.SubMatches(11)
    GetLogFromText.MessageID =
                        l_oMatch.SubMatches(8)
    GetLogFromText.SessionID =
                        l_oMatch.SubMatches(9)
    GetLogFromText.SetPropertyStream
                        l_oMatch.SubMatches(10)
End If
ErrorHandler:
End Function
```

Come si può osservare, l'algoritmo è molto semplice: si crea un textstream con la libreria FileSystemObject, sempre distribuita con lo Script Engine di Microsoft (libreria Microsoft Scripting Runtime), si legge il file di log attraverso il text stream e si sottopone tutto il testo dello stream al pattern regex che abbiamo già osservato. Una volta effettuato il matching, se vengono trovate delle occorrenze del pattern nella stringa e cioè se viene individuato il log item desiderato, viene creata un'istanza di LogItem popolata a partire da tutti i gruppi regex individuati (l_oMatch.Submatches(n)).
Il discorso si complica leggermente nel caso in cui si voglia estrarre l'elenco dei log item per range di date. In questo caso la regex deve insistere sulle occorrenze delle stringhe DATE e TIME del file di log:

```
LOGID="(.*)"\s*LEVEL=(\d{1})\s*DATETIME="(.*)-
(.*)"\r\n\s*CODE=\s*(.*)\r\n\s*TEXT=\s*(.*)\r\n\s*
SENDER=\s*(.*)\r\n\s*APPLICATION=\s*(.*)\r\n\s*
MESSAGEID=\s*(.*)\r\n\s*SESSIONID=\s*(.*)\r\n\s
*PROPERTIES=\s*(.*)\r\n\s*DEBUG_INFO=\s*(.*)
```

La particolarità di questo metodo rispetto al precedente è che non è possibile, con una sola regular expression, eseguire una ricerca fine su range di date, per cui procederemo con un metodo semimanuale: con la regex appena mostrata estraiamo tutti log item dal file di log e, per ciascuno di essi, verifichiamo se rientra nel range di date richiesto dalla GetLogs. Eccone l'implementazione del metodo:

```
Private Function ILogStorage_GetLogs(Optional ByVal
                StartDate As Date = #12:00:00 AM#, _
                        Optional ByVal
                EndDate As Date = #12:00:00 AM#, _
                        Optional ByVal
                LogStorageSettings As String = "", _
                        Optional ByVal
                ConsiderTime As Boolean = False) As
                        Amarcord.LogItems
    Set ILogStorage_GetLogs =
    GetLogsFromText(StartDate, EndDate, , ConsiderTime)
End Function
Public Const REGEX_GET_LOGITEMS As String = _
"LOGID=""(.*)""\s*LEVEL=(\d{1})\s*DATETIME=""(.*)-
(.*)""\r\n\s*CODE=\s*(.*)\r\n\s*TEXT=\s*(.*)\r\n\s*S
ENDER=\s*(.*)\r\n\s*APPLICATION=\s*(.*)\r\n\s*MES
SAGEID=\s*(.*)\r\n\s*SESSIONID=\s*(.*)\r\n\s*PROP
ERTIES=\s*(.*)\r\n\s*DEBUG_INFO=\s*(.*)"
Private Function GetLogsFromText(Optional ByVal
                        StartDate As Date = 0, _
                Optional ByVal EndDate As
                        Date = 0, _
                Optional ByVal LogFileName As
                        String = "", _
                Optional ByVal ConsiderTime
                        As Boolean = False) _
                As LogItems
    Dim l_oRegExp As VBScript_RegExp_55.RegExp
    Dim l_oMatchColl As
                VBScript_RegExp_55.MatchCollection
    Dim l_oMatch As VBScript_RegExp_55.Match
    Dim l_sTextFile As String
    Dim l_sText As String
    Dim l_oTextStream As Scripting.TextStream
    Dim l_oFSO As Scripting.FileSystemObject
    Dim l_oLogItem As LogItem
    Dim l_bDoIt As Boolean
    On Local Error GoTo ErrorHandler
    Set GetLogsFromText = New LogItems
    If LogFileName <> "" Then
        l_sTextFile = LogFileName
    Else
        l_sTextFile = m_sTextFile
```

```
End If
Set l_oFSO = New Scripting.FileSystemObject
Set l_oTextStream =
        l_oFSO.OpenTextFile(l_sTextFile, ForReading)
l_sText = l_oTextStream.ReadAll
Set l_oRegExp = New VBScript_RegExp_55.RegExp
l_oRegExp.Global = True
l_oRegExp.IgnoreCase = True
l_oRegExp.Pattern = REGEX_GET_LOGITEMS
Set l_oMatchColl = l_oRegExp.Execute(l_sText)
For Each l_oMatch In l_oMatchColl
    l_bDoIt = True
    If CLng(StartDate) > 0 Then
        l_bDoIt = l_bDoIt And
        (l_oMatch.SubMatches(2) >= Format(StartDate,
                                         "yyyyMMdd"))
    End If
    If CLng(EndDate) > 0 Then
        l_bDoIt = l_bDoIt And
        (l_oMatch.SubMatches(2) <= Format(EndDate,
                                         "yyyyMMdd"))
    End If
    If ConsiderTime Then
        If CLng(StartDate) > 0 Then
            l_bDoIt = l_bDoIt And
        (l_oMatch.SubMatches(3) >= Format(StartDate,
                                           "hhmmss"))
        End If
        If CLng(EndDate) > 0 Then
            l_bDoIt = l_bDoIt And
        (l_oMatch.SubMatches(3) <= Format(EndDate,
                                           "hhmmss"))
        End If
    End If
    If l_bDoIt Then
        Set l_oLogItem = New LogItem
        l_oLogItem.LogID = l_oMatch.SubMatches(0)
        l_oLogItem.LogDate =
                            l_oMatch.SubMatches(2)
        l_oLogItem.LogTime =
                            l_oMatch.SubMatches(3)
        l_oLogItem.LogLevel =
                            l_oMatch.SubMatches(1)
        l_oLogItem.MessageCode =
                            l_oMatch.SubMatches(4)
        l_oLogItem.MessageText =
                            l_oMatch.SubMatches(5)
        l_oLogItem.Sender = l_oMatch.SubMatches(6)
        l_oLogItem.Application =
                            l_oMatch.SubMatches(7)
        l_oLogItem.DebugInfo =
                            l_oMatch.SubMatches(11)
        l_oLogItem.MessageID =
                            l_oMatch.SubMatches(8)
        l_oLogItem.SessionID =
                            l_oMatch.SubMatches(9)
        l_oLogItem.SetPropertyStream
                            l_oMatch.SubMatches(10)
```

```
            GetLogsFromText.Add l_oLogItem
        End If
    Next
ErrorHandler:
    '
End Function
```

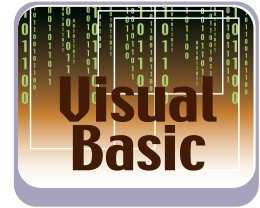
Quindi, come si sarà potuto notare, per ciascun log item, viene applicato il seguente controllo:

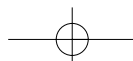
```
If CLng(StartDate) > 0 Then
    l_bDoIt = l_bDoIt And (l_oMatch.SubMatches(2) >=
                          Format(StartDate, "yyyyMMdd"))
End If
If CLng(EndDate) > 0 Then
    l_bDoIt = l_bDoIt And (l_oMatch.SubMatches(2) <=
                          Format(EndDate, "yyyyMMdd"))
End If
```

E il gioco è fatto. Una nota certamente interessante è quella sulle prestazioni. Se il dato sui tempi per il device basato su Access era ovvio e quello invece sulle query XPath su xml vi ha stupito per la sua rapidità, di sicuro il dato sui tempi di ricerca su file di testo attraverso regex vi sconvolgerà: meno di 160 millisecondi (sul notebook con AMD 64 da 2.0 GHz usato nelle prove) per i soliti 900 log item! Significa mediamente che è più lento di un DB o di xml solo del 25%. Incredibile, soprattutto considerando che la query regex per la GetLogs non è completamente filtrante come avviene per SQL o XPath, ma i dati estratti vanno elaborati successivamente da codice Visual Basic. Altri commenti sono inutili: vi lascio macerare nelle vostre riflessioni…

CONCLUSIONI

In questo articolo abbiamo progettato e realizzato un sistema di log dai tratti sicuramente innovativi perché basato su due caratteristiche fondamentali: la possibilità di recuperare le informazioni di log in un momento successivo attraverso ricerche fini e mirate e la possibilità di astrarsi dal medium fisico di memorizzazione del log attraverso un sistema di driver basati sul concetto di polimorfismo per interfaccia COM. Il tutto realizzato con tecnologie alla portata di tutti, a basso costo e in molti casi gratuite come Visual Basic 6, MSXML 3.0, ADO 2.6, Microsoft Scripting Engine e FileSystemObject: in pratica tecnologia Microsoft della precedente generazione… Provate a fare di meglio con .NET o con Java: magari ci riuscite, ma non dipenderà sicuramente da ciò che questi ambienti di sviluppo potranno darvi in più, ma solo dalla vostra fantasia. Fantasia che potrete mettere al lavoro anche per evoluzioni di questa idea.

Vito Vessia

L'AUTORE

Vito Vessia progetta e sviluppa applicazioni e framework in .NET, Visual Basic/COM e Delphi occupandosi degli aspetti architetturali. Scrive da anni per le principali riviste italiane di programmazione ed è autore del libro "PROGRAMMARE IL CELLULARE", (Hoepli), 2002, sulla programmazione dei telefoni cellulari connessi al PC con protocollo standard AT+. Può essere contattato tramite e-mail all'indirizzo vvessia@katamail.com.

I trucchi del mestiere

Tips & Tricks

Questa rubrica raccoglie trucchi e piccoli pezzi di codice, frutto dell'esperienza di chi programma, che solitamente non trovano posto nei manuali. Alcuni di essi sono proposti dalla redazione, altri provengono da una ricerca su Internet, altri ancora ci giungono dai lettori. Chi volesse contribuire, potrà inviare i suoi Tips&Tricks preferiti. Una volta selezionati, saranno pubblicati nella rubrica. Il codice completo dei tips è presente nel CD allegato nella directory *tips*\\ o sul Web all'indirizzo: cdrom.ioprogrammo.it.

CONVERTIRE DA FAHRENHEIT A CELSIUS

```
public static double CelsiusToFahrenheit
   (string temperatureCelsius)
{
   double celsius = System.Double.Parse (temperatureCelsius);
   return (celsius * 9 / 5) + 32;
}

public static double FahrenheitToCelsius
   (string temperatureFahrenheit)
{
   double fahrenheit = System.Double.Parse
                                   (temperatureFahrenheit);
   return (fahrenheit - 32) * 5 / 9;
}
```

COPIARE UN ARRAY?

```
using System;

public class CopyArray
{
   public static void Main()
   {
      int[]    integers = new int[] {5, 10, 15};
      double[] doubles  = new double[3];
      Array.Copy (integers, doubles, 2);
      Console.Write ("integers array elements:" );
      foreach (int integer in integers)
      {
        Console.Write("{0,3}", integer);
      }
      Console.Write ("\ndoubles  array elements:" );
      foreach (double duble in doubles)
      {
        Console.Write ("{0,3}", duble);
      }
      Console.WriteLine();
   }
```

PRENDERE UN ARGOMENTO DALLA LINEA DI COMANDO?

```
public static void Main (string[] args)
{
   if (args.Length > 0)
   {
      System.Console.WriteLine (args[0]);
      try
      {
//        long number = Int64.Parse(args[0]);
//        long number = Convert.ToInt64(s);
         long number = long.Parse (args[0]);

         System.Console.WriteLine ("Number = " + number);
      }
      catch (System.FormatException e)
      {
         System.Console.WriteLine ("FormatException" + e);
      }
      return;
   }

   System.Console.WriteLine ("Please enter a number");
}
```

RECUPERARE IL NOME NETBIOS DI UN COMPUTER

```
static string GetLocalHostName ()
{
   string netBiosName = System.Environment.MachineName;

   //return netBiosName;

   // Following method is deprecated
   // string dnsName =
   //   System.Net.Dns.GetHostByName("LocalHost").HostName;

   string dnsName = System.Net.Dns.GetHostName();
   return dnsName;
```

STAMPARE I NUMERI PRIMI FRA 1 E 1000?

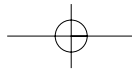
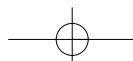
```
using System.Collections;
...
const int LAST_CANDIDATE = 1000;

int primes = 0;
BitArray candidates = new BitArray (LAST_CANDIDATE, true);

for (int i = 2; i < LAST_CANDIDATE; i++)
{
   if (candidates[i])
   {
      for (int j = i * 2; j < LAST_CANDIDATE; j += i)
         { candidates[j] = false; }
   }
}

for (int i = 1; i < LAST_CANDIDATE; i++)
{
   if (candidates[i])
   {
      primes++;
      Console.Out.WriteLine (i);
   }
}

Console.Out.WriteLine
   ("\n" + primes + " primes found in the range 2-" +
                                        LAST_CANDIDATE);
```

CANCELLARE LA CACHE DI INTERNET EXPLORER

```
using System.IO;
...
void clearIECache()
{
   ClearFolder (new DirectoryInfo (Environment.GetFolderPath
      (Environment.SpecialFolder.InternetCache)));
}

void ClearFolder (DirectoryInfo folder)

{
   foreach (FileInfo file in folder.GetFiles())
      { file.Delete(); }
   foreach (DirectoryInfo subfolder in folder.GetDirectories())

      { ClearFolder(subfolder); }
}

public static void Main( )
{

   new Test().clearIECache ();
}
```

IL CD È INSERITO?

```
using System;
using System.Management;

class App
{
  public static void Main()
  {
    SelectQuery query = new SelectQuery( "select * from
                          win32_logicaldisk where drivetype=5" );
    ManagementObjectSearcher searcher = new
                             ManagementObjectSearcher(query);

    foreach( ManagementObject mo in searcher.Get() )
    {
      // If both properties are null I suppose there's no CD
      if( ( mo["volumename"] != null ) ||
                       ( mo["volumeserialnumber"] != null ) )
      {
        Console.WriteLine( "CD is named: {0}",
                                        mo["volumename"] );
        Console.WriteLine( "CD Serial Number: {0}",
                                 mo["volumeserialnumber"] );
      }
      else
      {
        Console.WriteLine( "No CD in Unit" );
      }
    }
    // Here to stop app from closing
    Console.WriteLine( "\nPress Return to exit." );
    Console.Read();
  }
}
```

CALCOLARE L'ETÀ DI UNA PERSONA?

```
Public Function GetAge(ByVal Birthdate As System.DateTime, _
    Optional ByVal AsOf As System.DateTime = #1/1/1700#) _
    As String

        'Don't set second parameter if you want Age as of today

        'Demo 1: get age of person born 2/11/1954
        'Dim objDate As New System.DateTime(1954, 2, 11)
        'Debug.WriteLine(GetAge(objDate))

        'Demo 1: get same person's age 10 years from now
        'Dim objDate As New System.DateTime(1954, 2, 11)
        'Dim objdate2 As System.DateTime
        'objdate2 = Now.AddYears(10)
```

```
        'Debug.WriteLine(GetAge(objDate, objdate2))

        Dim iMonths As Integer
        Dim iYears As Integer
        Dim dYears As Decimal
        Dim lDayOfBirth As Long
        Dim lAsOf As Long
        Dim iBirthMonth As Integer
        Dim iAsOFMonth As Integer

        If AsOf = "#1/1/1700#" Then
            AsOf = DateTime.Now
        End If
        lDayOfBirth = DatePart(DateInterval.Day, Birthdate)
        lAsOf = DatePart(DateInterval.Day, AsOf)

        iBirthMonth = DatePart(DateInterval.Month, Birthdate)
        iAsOFMonth = DatePart(DateInterval.Month, AsOf)

        iMonths = DateDiff(DateInterval.Month, Birthdate, AsOf)

        dYears = iMonths / 12

        iYears = Math.Floor(dYears)

        If iBirthMonth = iAsOFMonth Then
            If lAsOf < lDayOfBirth Then
                iYears = iYears - 1
            End If
        End If

        Return iYears
    End Function
```

CHIAMARE UN'API

```
Imports System.Runtime.InteropServices

'Required in this example and any API function which
'use a string buffer.  Provides the StringBuilder class

Imports System.Text

'Put these declarations right under the class declaration
'(e.g., in a Form, right under Public Class Form1)

  <DllImport("KERNEL32.DLL",
                              EntryPoint:="GetSystemDirectoryW", _
      SetLastError:=True, CharSet:=CharSet.Unicode, _
      ExactSpelling:=True, _
      CallingConvention:=CallingConvention.StdCall)> _
      Public Shared Function GetSystemDirectory(ByVal Buffer _
        As StringBuilder, ByVal Size As Integer) As Long

        ' Leave function empty - DLLImport attribute
        ' forces calls to GetSystemDirectory to
        ' be forwarded to GetSystemDirectory in KERNEL32.DLL
    End Function
```

```
     Public Const MAX_PATH As Integer = 256
```

e per usarla...

```
         Dim s As New StringBuilder(MAX_PATH)

         GetSystemDirectory(s, MAX_PATH)
         msgbox(s.ToString(), , "System Directory")

Come posso sapere se esiste un'istanza di un'applicazione?
   Public Sub CheckForExistingInstance()
         'Get number of processes of you program
         If Process.GetProcessesByName _
           (Process.GetCurrentProcess.ProcessName).Length > 1 Then

         MessageBox.Show _
         ("Another Instance of this process is already running", _
             "Multiple Instances Forbidden", _
              MessageBoxButtons.OK, _
             MessageBoxIcon.Exclamation)
            Application.Exit()
         End If
     End Sub
```

CONVERTIRE UN'IMMAGINE BITMAP?

```
Imports System.IO.Path
Imports System.Drawing.Imaging
   Public Function ConvertBMP(ByVal BMPFullPath As String, _
    ByVal imgFormat As ImageFormat) As Boolean
        Dim bAns As Boolean
        Dim sNewFile As String

        Try
            'bitmap class in system.drawing.imaging
            Dim objBmp As New Bitmap(BMPFullPath)

            'below 2 functions in system.io.path
            sNewFile = GetDirectoryName(BMPFullPath)
            sNewFile &=
                           GetFileNameWithoutExtension(BMPFullPath)

            sNewFile &= "." & imgFormat.ToString
            objBmp.Save(sNewFile, imgFormat)
            bAns = True 'return true on success
        Catch
            bAns = False 'return false on error
        End Try
        Return bAns
    End Function
    'USAGE
    'ConvertBMP("C:\test.bmp", ImageFormat.Jpeg)
    'ConvertBMP("C:\test.bmp", ImageFormat.Emf)
    'ConvertBMP("C:\test.bmp", ImageFormat.Exif)
    'ConvertBMP("C:\test.bmp", ImageFormat.Gif)
    'ConvertBMP("C:\test.bmp", ImageFormat.Icon)
```

072-076 29-06-2007 16:37 Pagina 79


```
'ConvertBMP("C:\test.bmp", ImageFormat.MemoryBmp)
'ConvertBMP("C:\test.bmp", ImageFormat.Png)
'ConvertBMP("C:\test.bmp", ImageFormat.Tiff)
'ConvertBMP("C:\test.bmp", ImageFormat.Wmf)
```

CREARE UN DATABASE ACCESS A RUNTIME?

```
Dim cat As New ADOX.Catalog
Dim bAns As Boolean
Try
  Dim sCreateString As String
  Dim databasePath As String
  databasePath = Application.StartupPath() & "\Export.mdb"
  sCreateString = _
  "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" &
                                                databasePath
   cat.Create(sCreateString)
   bAns = True
Catch ex As Exception
      bAns = False
      MessageBox.Show(ex.ToString)
Finally
      cat = Nothing
   End Try
```

DISABILITARE IL TASTO INVIO

```
<script language=javascript type=text/javascript>
<!-- Script courtesy of http://www.web-source.net - Your Guide
                   to Professional Web Site Design and Development
function stopRKey(evt) {
   var evt = (evt) ? evt : ((event) ? event : null);
   var node = (evt.target) ? evt.target : ((evt.srcElement) ?
                                         evt.srcElement : null);
   if ((evt.keyCode == 13) && (node.type=="text")) {return
                                                    false;}
}

document.onkeypress = stopRKey;
-->
</script>
```

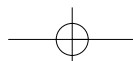
AGGIUNGERE UN BOTTONE DEL TIPO "FAI DI QUESTA PAGINA LA TUA HOME

```
<FORM>
<INPUT TYPE="button" VALUE="Make This Site Your Home
   Page" onClick="this.style.behavior='url(#default#homepage)';
     this.setHomePage('Page URL beginning with http:// here');">
</FORM>
```

AGGIUNGERE UN'OROLOGIO NELLA STATUS BAR

```
<script Language="JavaScript">
<!--
var timerID = null;
var timerRunning = false;

function stopclock (){
if(timerRunning)
clearTimeout(timerID);
timerRunning = false;
}

function showtime () {
var now = new Date();
var hours = now.getHours();
var minutes = now.getMinutes();
var seconds = now.getSeconds()
var timeValue = "" + ((hours >12) ? hours -12 :hours)

timeValue += ((minutes < 10) ? ":0" : ":") + minutes
timeValue += ((seconds < 10) ? ":0" : ":") + seconds
timeValue += (hours >= 12) ? " P.M." : " A.M."
window.status = timeValue;

// you could replace the above with this
// and have a clock on the status bar:
//
timerID = setTimeout("showtime()",1000);
timerRunning = true;
}

function startclock () {
// Make sure the clock is stopped
stopclock();
showtime();
}

// -->
</script>
```

RIDIMENSIONARE UN ARRAY

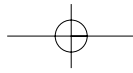
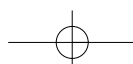
```
/**
* Reallocates an array with a new size, and copies the contents
* of the old array to the new array.
* @param oldArray  the old array, to be reallocated.
* @param newSize   the new array size.
```

```
* @return          A new array with the same contents.
*/
private static Object resizeArray (Object oldArray, int newSize)
                                                              {
   int oldSize = java.lang.reflect.Array.getLength(oldArray);
   Class elementType =
                           oldArray.getClass().getComponentType();
   Object newArray = java.lang.reflect.Array.newInstance(
         elementType,newSize);
   int preserveLength = Math.min(oldSize,newSize);
   if (preserveLength > 0)
      System.arraycopy (oldArray,0,newArray,0,preserveLength);
   return newArray; }


// Test routine for resizeArray().
public static void main (String[] args) {
   int[] a = {1,2,3};
   a = (int[])resizeArray(a,5);
   a[3] = 4;
   a[4] = 5;
   for (int i=0; i<a.length; i++)
      System.out.println (a[i]); }
```

RIDIMENSIONARE UN ARRAY BIDIMENSIONALE?

```
  int a[][] = new int[2][3];
  //...
  a = (int[][])resizeArray(a,20);
    // new array is [20][3]
  for (int i=0; i<a.length; i++) {
     if (a[i] == null)
        a[i] = new int[30];
     else a[i] = (int[])resizeArray(a[i],30); }

   // new array is [20][30]
```

COMPARARE UN ELENCO DI PREZZI

```
public class BestPrice
{  public static void main(String[] args)
   {  final int DATA_LENGTH = 1000;
      double[] data = new double[DATA_LENGTH];
      int dataSize = 0;

      // read data

      ConsoleReader console = new ConsoleReader(System.in);

      boolean done = false;
      while (!done)
      {  System.out.println("Enter price, 0 to quit:");
         double price = console.readDouble();
         if (price == 0) // end of input
            done = true;
         else if (dataSize < data.length)
         {  // add price to data array

            data[dataSize] = price;
            dataSize++;
         }
         else // array is full

         {  System.out.println("Sorry, the array is full.");
            done = true;
         }
      }

      // compute lowest price

      if (dataSize == 0) return; // no data
      double lowest = data[0];
      for (int i = 1; i < dataSize; i++)
         if (data[i] < lowest) lowest = data[i];

      // print out prices, marking the lowest one

      for (int i = 0; i < dataSize; i++)
      {  System.out.print(data[i]);
         if (data[i] == lowest)
            System.out.print(" <-- lowest price");
         System.out.println();
      }
   }
}
```

CONVERTIRE UNA STRINGA IN UN NUMERO

```
public class StringToNum{

  static long stringToNum(String rep, int base){
    long retVal=0;
    long placeVal=1;
    for (int i=rep.length()-1; i>=0; i--){
      retVal=retVal+placeVal* (rep.charAt(i) - '0');
      placeVal=placeVal*base;
    }
    return retVal;
  }

  public static void printConvert(String rep, int base){
    System.out.println(rep + " in base " + base +  " = " +
                     stringToNum(rep,base));
  }

  public static void main(String args[]){
    printConvert("11011011",2);
    printConvert("7103",8);
    printConvert("123",10);
    printConvert("123704",8);
      }

}
```

FACADE: UN SISTEMA A COLPO D'OCCHIO

PROPRIO COME FACCIAMO CON I SINGOLI OGGETTI, A VOLTE VOGLIAMO NASCONDERE LA COMPLESSITÀ DI UN INTERO SISTEMA DIETRO UN'INTERFACCIA SEMPLICE. FARLO NON È DIFFICILE, BASTA APPLICARE IL PATTERN FACADE. ECCO COME...

È un mestiere ingrato, quello di programmatore. Certe volte sembra che i tuoi stessi colleghi ce la mettano tutta per renderti la vita difficile. L'azienda di Beatrice, la brava programmatrice, sta costruendo un sistema di prenotazioni ferroviarie. Ci lavorano tre team separati: il team rosso, dove lavora Beatrice, si occupa del server che fornisce tutti i servizi importanti, come cercare un posto o prenotarlo; il team blu sta costruendo un'interfaccia web per mettere il sistema su Internet; il team verde sta lavorando ad un client grafico per le agenzie di viaggio.
Beatrice ha lavorato per tutto il mese per sviluppare un server flessibile e potente, in modo che il team blu e quello verde avessero tutte le funzionalità che servono ai loro client. E cosa fanno, questi ingrati del team verde? Anziché ringraziare Beatrice per l'ottimo lavoro svolto, segnalano la loro disapprovazione in tutti i modi: mail minatorie, mortaretti sotto la sedia, persino un biglietto appeso ad un coltello conficcato nella porta dell'ufficio. Il biglietto, composto con lettere ritagliate dai giornali, è fin troppo esplicito:

"Il tuo sistema è una schifezza. Cambialo. Se dovesse restare com'è... Be', si sa che i dischi rigidi, a volte, possono smagnetizzarsi. Ci siamo capiti. I tuoi amici del Team Verde".

Questo è troppo, pensa Beatrice. E' ora di trovare una soluzione.

UN LAVORO BEN FATTO

Il sistema di Beatrice è un bellissimo programma Ruby scritto secondo tutti i crismi della programmazione object-oriented. Ciascuna entità (treni, orari, utenti) è accuratamente incapsulata nella sua bella classe. Ad esempio, vediamo come fa un client a cercare posti liberi su un treno che viaggia da Bologna a Lecce il 30 agosto 2007.
Per accedere al sistema, il client deve chiamare il metodo *login()* sulla classe *Server*. Questa è una normale chiamata locale, ma un sistema remoto può usare il metodo *connect_to()* per ottenere un oggetto locale che simula un server e invia le chiamate al vero server attraverso la rete (per chi ha letto gli articoli dei mesi scorsi, questo è un esempio del pattern Proxy). Ad esempio, ecco un client che si collega ad un server sulla porta 8081 dello stesso computer:

```
server = connect_to("localhost", 8081)
```

Ora che ha un riferimento locale al server, il client può fare il login:

```
usr = server.login("Paolo_Perrotta",
                   "laMiaBellaPassword")
puts usr
```

Il metodo *login()* restituisce un oggetto *User*, che contiene un codice identificativo e i dati dell'utente. Questo oggetto viene restituito via rete al client, che può usarlo nelle successive chiamate per identificarsi senza ripetere l'autenticazione. L'istruzione puts stampa un oggetto sullo schermo, e nel caso dell'utente dà un risultato come questo:

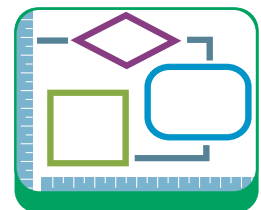
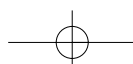
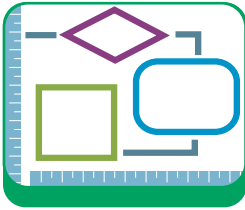
```
Paolo Perrotta (id:1234)
```

Ora il client può chiedere gli orari dei treni per il giorno 30 agosto 2007 con una chiamata come questa:

```
timeline = server.timeline_for(user, 30, 8, 2007)
```

Il risultato è un oggetto *Timeline* che contiene riferimenti a tutti i treni di un dato giorno. La *Timeline* contiene solo i riferimenti, non i treni veri e propri. Dato che la Timeline deve viaggiare sulla rete, non avrebbe senso spedire un oggetto enorme con tutti gli orari. Invece, il client può interrogare la *Timeline* per chiederle solo i treni che viaggiano da Lecce a Bologna:

```
interesting_trains = timeline.on_route("Bologna",
                                       "Lecce")
puts interesting_trains
```

Il nostro client ha trovato tre oggetti di classe *Train* (abbiamo usato nomi di fantasia per evitare che le Ferrovie ci facciano causa):

081-083 29-06-2007 16:38 Pagina 82


```
Freccia delle paludi
Euroflop
Il Tartarugone
```

Ora che il client ha un array di treni, può esaminarli uno per uno per chiedere a ciascuno se ci sono posti liberi:

```
available_trains = interesting_trains.find_all {|train|
                                        train.free_seats? }
puts available_trains
```

La prima riga del codice qui sopra usa quello che in Ruby si chiama un *blocco*. E' quella roba tra parentesi graffe. Non abbiamo spazio in questo articolo per spiegare in dettaglio come funzionano i blocchi, ma potete vederli come un pezzo di codice che viene passato al metodo *find_all()*. Abbiamo anche omesso le parentesi tonde vuote nella chiamata al metodo *find_all()* - in Ruby, omettere le parentesi nelle chiamate ai metodi è sempre legale. Il codice qui sopra chiama il metodo *find_all()* della classe *Array*, che seleziona tutti gli elementi dell'array per i quali il blocco restituisce *true*. Quindi in questo caso il blocco contiene la condizione per decidere se un elemento deve o no fare parte del risultato. Questa condizione è che ci siano posti liberi sul treno, cosa che possiamo verificare chiamando il metodo *free_seats?()* della classe *Train*. Notate che il punto interrogativo fa parte del nome del metodo, e che anche in questo caso abbiamo omesso le parentesi tonde nella chiamata.
Il risultato finale è la lista dei treni che hanno posti vuoti:

```
Freccia delle paludi
Il Tartarugone
```

(Ovviamente questa è solo una simulazione. Nella realtà, tutti i treni che viaggiano da Bologna a Lecce il 30 agosto sarebbero interamente prenotati da maggio).
Tutto questo è solo un esempio, ma il sistema di Beatrice permette al client di fare ricerche molto più sofisticate.
Ad esempio, si può controllare quali treni sono disponibili nei giorni precedente e successivo ad una certa data. Se invece il client trova un posto che gli piace, può cercare di prenotarlo. Per fare questo deve prima guardare nello *User* per scoprire qual è il suo sistema di pagamento preferito... Ma questa è un'altra storia. Nel nostro esempio, ci resta solo da disconnetterci elegantemente per non avere problemi al prossimo login:

```
server.logout(user)
```

Tutto molto bello e ordinato. Ma allora, perché il team verde è così furioso?

POCO CODICE, SIAMO FACADE

I metodi di una Facade contengono pochissimo codice, e delegano tutta la logica alle classi più interne del sistema. Se la vostra Facade contiene troppo codice, dovreste riflettere se sia il caso di spostare questa logica fuori dalla Facade.

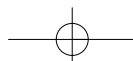
IL TEAM VERDE PARLA

Decisa a risolvere la questione da persona civile, Beatrice chiede ed ottiene di parlare con il capo del team verde. Durante l'incontro (che si svolge in territorio neutrale, nella sala mensa deserta a mezzanotte), Beatrice ottiene presto qualche commento costruttivo sul suo server.
"Per prima cosa, questo server è troppo complicato", sostiene il Team Leader Verde. "Tutti questi oggetti e questi pattern saranno anche fighi, ma la verità è che noi non ci capiamo niente. Il novantanove per cento delle volte, quello che dobbiamo fare sono operazioni semplicissime. Ad esempio vogliamo verificare se ci sono posti disponibili su un qualsiasi treno che viaggia tra due determinate stazioni in un dato giorno. Per un'operazione banale come questa, ci tocca fare almeno quattro chiamate al server. Dobbiamo prima fare il login, poi chiedere la *Timeline*, poi usare questa per selezionare i treni tra due stazioni, e infine esaminarli uno per uno. E dobbiamo anche ricordarci di fare il logout. E poi ovviamente poi si deve prenotare il treno, e anche per quello dobbiamo fare tre o quattro chiamate".
"La gran parte delle operazioni che facciamo richiede tutta una serie di passaggi come questi", continua il Capo Verde. "Ci basta sbagliare un passaggio per causare un bug. E come se non bastasse, ogni operazione può lanciare diverse eccezioni. Ma noi gestiamo tutte le eccezioni nello stesso modo, quindi anche in questo caso non ci serve tutta questa complessità".
"Ma la cosa più seccante", continua il lamentoso collega di Beatrice, "è che le prestazioni del sistema sono pessime. Con tutti questi dati che vanno avanti e indietro per la rete, e tutte queste chiamate per ogni singola operazione, il sistema diventa lentissimo. Poi i clienti delle agenzie se la prendono con noi, quando noi non abbiamo alcuna colpa. E' tutta colpa del tuo server!"
Al posto di Beatrice, un altro sviluppatore perderebbe la calma. Ma la nostra amica è una persona che sa fare il suo mestiere, e capisce le ragioni del suo molesto collega. La pulizia e l'eleganza del server si traduce in frustrante complessità a carico del client. O il sistema è flessibile, o è facile da usare.
Per fortuna, Beatrice sa già come uscire da questo dilemma e dare al team verde quello che chiede senza modificare una riga del suo sistema.

UNO STRATO DI SEMPLICITÀ

Tutto quello che Beatrice deve fare è aggiungere una nuova interfaccia al sistema. Ad esempio, il client del Team Verde potrebbe chiamare una classe fatta così:

```
class BookingSystem
  def self.find_available_trains(server_address,
```

081-083 29-06-2007 16:39 Pagina 83


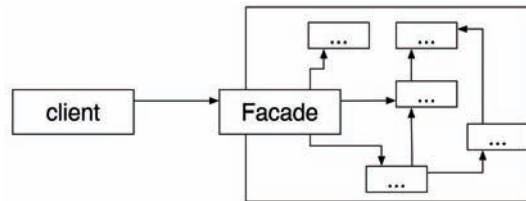
```
    server_port, username, password, day, month, year,
                                              from, to)
  server = connect_to(server_address, server_port)
  user = server.login(username, password)
  timeline = server.timeline_for(user, day, month,
                                                  year)
  interesting_trains = timeline.on_route :from =>
                                         from, :to => to
  available_trains = interesting_trains.find_all
                          {|train| train.free_seats? }
  server.logout(user)
  return available_trains
 end
end
```

La parola chiave self indica che il metodo *find_available_trains()* è un metodo di classe (quello che in Java si chiamerebbe un "metodo statico"). Quindi può essere chiamato direttamente sulla classe, senza creare un oggetto. In pratica è una funzione che permette di scoprire, con una sola chiamata, tutti i treni che hanno posti liberi e viaggiano tra due stazioni in un dato giorno. Questo metodo si occupa da solo del login, del logout e di tutte le operazioni che ci sono in mezzo. Il client può chiamarlo così:

```
puts BookingSystem.find_available_trains("localhost",
 8081, "Paolo_Perrotta", "laMiaPasword", 30, 8, 2007,
                                   "Bologna", "Lecce")
```

La stessa classe potrebbe avere un metodo di nome *book()* per prenotare un treno. Non si potrebbe scrivere un unico metodo che mette insieme *find_available_trains() e book()*? Un metodo simile non è una buona idea, semplicemente perché il client non ne ha bisogno. Prima di prenotare un posto, tutti i viaggiatori che si presentano in agenzia vogliono conoscere un elenco di posti tra i quali scegliere. Quindi questa interfaccia ha esattamente la "granularità" richiesta dal client. Non è a grana sottile come l'originale sistema a oggetti, che richiedeva al client una lunga serie di chiamate per ciascuna semplice operazione; ma non è nemmeno a grana così grossa da limitare il client. E se proprio i nostri minacciosi amici del team verde volessero aggiungere qualche funzionalità dell'ultimo momento, potrebbero sempre girare intorno alla classe *BookingSystem* e usare direttamente le normali classi del sistema – per una volta, non gli farà male.

IL PATTERN FACADE

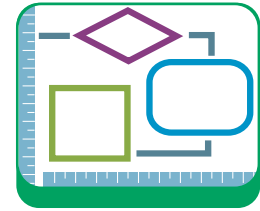
La classe *BookingSystem* è un esempio del design pattern Facade. Il nome "*Facade*" si legge "fasàd", alla francese, e significa "facciata". L'idea alla base del pattern è semplice: se hai un sistema complicato, puoi metterci davanti una facciata che lo semplifica. Una Facade è sottile come le case finte che si usano per girare i film western: contiene pochissimo codice, di solito una semplice sequenza di chiamate alle classi più interne del sistema.

Una Facade è utile perché semplifica un sistema complesso senza intaccarne la potenza. Un client può chiamare le operazioni della Facade senza capire come interagiscono tutti gli elementi del sistema. Quando questo non basta, il client può sempre decidere di aggirare la facciata e raggiungere direttamente il cuore del sistema.
Il pattern Facade diminuisce anche l'accoppiamento tra il sistema e i suoi client. Se tutte le comunicazioni passano attraverso la Facade, è facile modificare il sistema senza che il client se ne accorga.
Inoltre, se il client è un sistema remoto, la Facade riduce il numero di chiamate al server. Infatti, pochi giorni dopo l'introduzione della Facade, il client sviluppato dall'azienda di Beatrice per le agenzie di viaggio è diventato due volte più veloce. Due piccioni con una fava, pensa Beatrice. Finché una mattina non trova sulla scrivania un mazzo di fiori e un bigliettino scritto con ritagli di giornale:

"Che ne diresti di cenare insieme una di queste sere? Il capo del Team Verde".

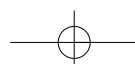
Ecco fatto, pensa sconsolata Beatrice. Appena risolto un problema, subito ne nasce un altro.

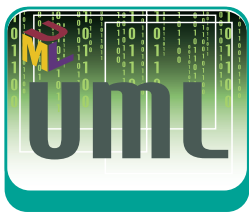
CONCLUSIONI

A volte la flessibilità non va d'accordo con la semplicità. Quando volete che il vostro sistema sia flessibile e semplice allo stesso tempo, potete usare tutti i pattern che volete all'interno del sistema, e poi esporlo attraverso una Facade - un'interfaccia più rozza, ma anche più semplice.
Quello di questo mese è stato un pattern rilassante e tecnicamente semplice. Ma nei prossimi numeri di IoProgrammo troverete pane più duretto per i vostri denti. A presto!

Paolo Perrotta

NOTA

UNA FACCIATA IN JAVA

In questo articolo abbiamo usato Ruby. Cosa cambia se usiamo Java o un altro linguaggio "staticamente tipato"? Quasi niente, in realtà. Anche se il compito di una Facade è quello di "fare da interfaccia", l'implementazione è di solito un'unica classe concreta. Ma potete anche costruire una Facade come un'interface Java con diverse implementazioni, in modo da accedere con la stessa interfaccia a sistemi diversi.

MODELLIAMO IL SOFTWARE CON UML

UTILIZZIAMO UML PER PROGETTARE UN SOFTWARE PER LA GESTIONE DELLA NOSTRA SQUADRA DI CALCIO. TEAM2007 È UN'APPLICAZIONE CHE NON PUÒ MANCARE NEL COMPUTER DI OGNI PRESIDENTE ATTENTO ALLA TECNOLOGIA

Nel numero precedente abbiamo iniziato questo breve corso di UML studiando la realizzazione di un progetto concreto: un'applicazione per la gestione della propria squadra di calcio che si chiama Team2007. Nella puntata precedente sono stati elencati i diagrammi più importanti di UML. Sono stati descritti gli Use Case e sono stati introdotti i diagrammi di classe. In questa puntata verrà completata la descrizione di questi ultimi. Come nel numero scorso utilizzeremo l'applicazione di modellazione Argo UML (http://argouml.tigris.org/), descrivendone anche il suo utilizzo pratico.

UN VIAGGIO IN UML

Nella mitologia greca, Argo era la nave che portò Giasone e gli argonauti alla conquista del vello d'oro. Infatti l'icona del programma Argo UML, almeno su Mac OS X, è una simpatica barchetta di carta. Il programma è organizzato su una finestra singola (**Figura 1**), che è suddivisa in diversi pannelli:

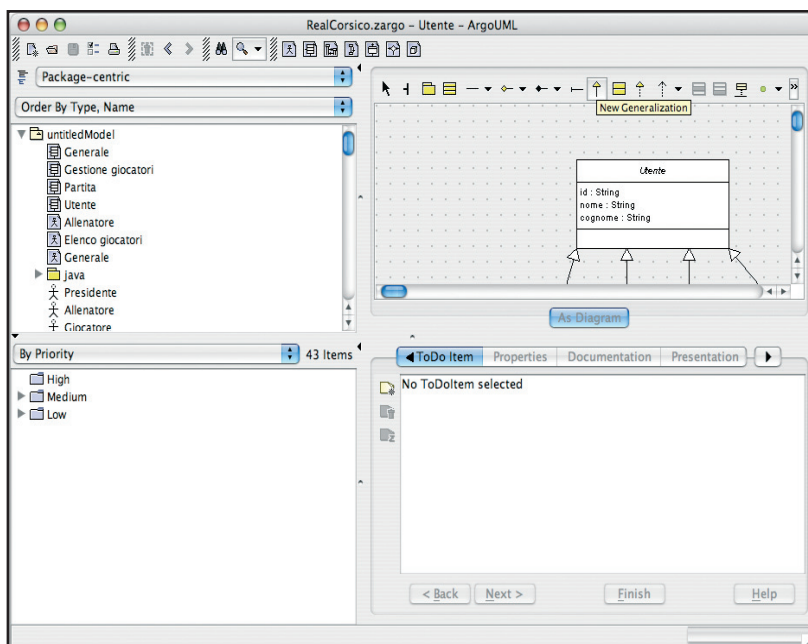



Fig. 1: Finestra principale del programma Argo UML.

- in alto a sinistra è presente una outline con l'elenco dei diagrammi e delle entità definite nel modello che si sta realizzando. Tramite le due caselle di selezione è possibile scegliere il tipo di vista e l'ordinamento desiderati. La prima impostazione permette di scegliere se incentrare la visione sui package, sulle classi, sulle associazioni delle classi o sugli stati. In questo modo è possibile utilizzare UML non solo per la progettazione del software, ma anche per la modellazione funzionale. La seconda casella di selezione permette di ordinare per tipo e nome o solo per nome. La vista di questo pannello è gerarchica. Dove ha senso è quindi possibile espandere un nodo per visualizzare quello che esso contiene. Per esempio, nella visualizzazione basata su package è possibile espandere un package per visualizzarne il suo contenuto, classi comprese.

- Nel riquadro sottostante è presente un elenco delle critiche che Argo UML fa al nostro codice. Ebbene sì: il programma non si limita a permettere all'utente di disegnare diagrammi, ma entra anche nel merito degli stessi e li valuta sulla base di un insieme di regole predefinite. Le critiche possono essere visualizzate per priorità, decisione, obiettivo e altro. Interessante la vista per "conoscenza", che suddivide tra problemi di correttezza, completezza, consistenza, sintassi, semantica ed altro. Questa funzionalità si controlla con la voce di menu Critique. Da qui è possibile disabilitare la funzione, impostare la priorità di ciascuna regola oppure visualizzare la configurazione del sistema di controllo.

- Il riquadro in alto a destra è quello più importante dell'applicazione, perché contiene l'area di disegno che si utilizza per tracciare e visualizzare i diagrammi. Per default è caratterizzata da uno sfondo con una griglia formata da puntini, che aiuta nell'allineamento e dimensionamento degli elementi grafici. Sopra l'area di disegno è presente una barra degli strumenti che si modifica in funzione del tipo di dia-

084-087 29-06-2007 16:46 Pagina 85

gramma che si sta disegnando.

- Il riquadro in basso a destra contiene invece una serie di schede di approfondimento relative all'elemento attualmente selezionato. Per esempio, selezionando una classe vengono visualizzate le schede ToDo, Properties, Documentation, Presentation, Source, Constraints, Stereotype, Tagged Values e Checklist. Ciascuna di queste consente di manipolare determinati aspetti dell'elemento attivo. Per esempio, con la scheda Properties è possibile impostare tutti i modificatori di classe, la visibilità, gli attributi, le operazioni e molto altro. La scheda Source consente invece di visualizzare la classe sottoforma di codice Java.

Argo UML è abbastanza completo, ma i suoi menu di sistema non sono sovraccaricati di funzioni inutili e complesse. Sotto il menu File si trovano le classiche funzionalità di gestione dei file, come l'apertura e il salvataggio dei progetti, l'importazione e l'esportazione, la stampa e l'esportazione sotto forma di immagini grafiche e l'impostazione delle proprietà del modello su cui si sta lavorando. Il menu Edit contiene funzioni di selezione e configurazione delle viste e delle impostazioni generali del programma. Da qui è possibile anche cancellare un elemento, in due modi diversi. Il primo è quello di rimuovere l'elemento solo dal diagramma. In questo caso l'elemento rimane nel modello, e viene elencato nel primo pannello del programma. Sarà quindi possibile riutilizzarlo in seguito. La seconda opzione è quella di rimuovere l'elemento dal modello. In tal caso questo viene rimosso completamente, e sparisce da qualsiasi diagramma che lo conteneva.
Il menu View consene di saltare a un determinato diagramma, modificare lo zoom, la griglia o l'allineamento. C'è anche la funzione "Trova", che è discretamente potente (**Figura 2**). Questa permette di individuare i diagrammi che contengono determinati elementi, specificando:

- il tipo (classe, interfaccia, attori ecc.);
- il nome da cercare;
- il diagramma;
- il perimetro di ricerca.

Il risultato della ricerca viene visualizzato nella finestra di ricerca stessa. È possibile fare doppio clic sugli elementi trovati per visualizzarli nel pannello del diagramma.
Tornando al menu, sotto Create sono disponibili le voci di menu per creare i diversi diagrammi supportati dal programma.
Sotto Arrange sono disponibili invece le funzioni per allineare, distribuire e riorganizzare gli elementi di un diagramma o per eseguire il layout automatico.
Infine, il menu Generation consente di produrre il codice sorgente Java relativo alle classi presenti nel modello. Si noti che ad oggi Argo UML è limitato al linguaggio di SUN Microsystems, ma che è progettato per poter essere esteso in futuro ad altri linguaggi di programmazione.

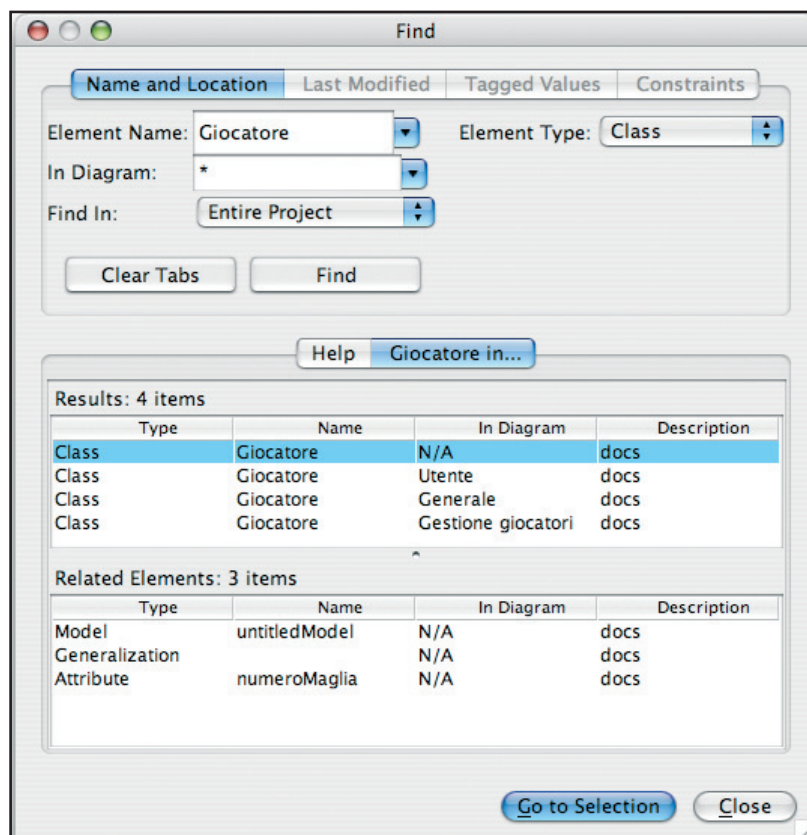



Fig. 2: Funzione di ricerca di Argo UML.

EREDITARIETÀ

Un aspetto fondamentale della programmazione orientata agli oggetti è la possibilità di creare nuove classi estendendo quelle esistenti. UML ovviamente dispone delle convenzioni necessarie a rappresentare questo aspetto. Nella fattispece, si utilizza una freccia che termina con una punta chiusa e vuota.
Nella Figura 3 è presente un esempio tratto dal progetto di Team2007. Come si nota, è presente la classe Utente, da cui derivano Presidente, Allenatore, Giocatore e Tifoso. A sua volta, la classe Giocatore è la superclasse di Portiere. Questa gerarchia di classi descrive le tipologie di utenti, o ruoli, che sono gestiti dall'applicazione.
Creare questo diagramma con Argo UML è molto semplice. È sufficiente selezionare Create | New Class Diagram e poi trascinare dalla outline presente nel pannello a sinistra le classi che interessano. Se queste sono già state messe in relazione in altri diagrammi, l'informazione relativa è già memorizzata nel modello, quindi viene riportata in automatico nel nuovo diagramma. Per esempio, si ipotizzi di

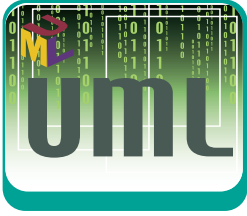
Il sito ufficiale di UML è raggiungibile all'indirizzo www.uml.org, dove è possibile accedere a documentazione e approfondimenti relativi a questa interessante tecnologia di modellazione e rappresentazione grafica.

modificare il file di progetto che ha originato la Figura 3. Se si crea un nuovo diagramma di classe e si trascinano nell'area di disegno le due classi Utente e Giocatore, queste vengono già rappresentate con un legame di ereditarietà.
Se invece il modello non presenta le classi che si vuole rappresentare nel diagramma che si sta realizzando, è possibile crearle facendo clic sull'icona della classe nella barra degli strumenti dell'area di disegno e poi facendo clic in un qualsiasi punto

Fig. 3: Ereditarietà.

della stessa. In questo modo viene creata una classe vuota, che ovviamente è necessario compilare con gli attributi e le operazioni che si ritiene opportune. Per fare questo si utilizza il pannello inferiore destro (quello delle proprietà dell'oggetto attivo).

Se interessa approfondire UML e il suo rapporto con i Design Pattern è possibile consultare il testo: Massimiliano Bigatti, "UML e Design Pattern. Notazioni grafiche e soluzioni per la progettazione", Hoepli Editore, ISBN: 978-88-203-3657-8

MODIFICA DI UNA CLASSE

Per impostare gli elementi di una classe è possibile fare clic direttamente nell'area che si vuole modificare: sia essa il nome della classe, l'elenco delle proprietà o delle operazioni. In alternativa, per cambiare il nome della classe è possibile utilizzare la casella Name del pannello delle proprietà.
Quest'ultimo contiene anche l'elenco degli attributi e delle operazioni, la cui funzionalità è però limitata. È possibile infatti solo modificare l'ordine degli elementi oppure modificarli. Questo secondo caso si verifica quando si fa clic sul nome dell'operazione o dell'attributo. Questi infatti sono rappresentati come collegamenti ipertestuali. Se selezionati modificano l'intero pannello delle proprietà, focalizzandone il contenuto sull'elemento selezionato e non più sulla classe che si stava modificando.
Una volta impostati gli elementi di una classe è possibile impostare le relazioni di ereditarietà. Per fare questo è possibile agire in due modi. Il primo inizia con la selezione della classe attraverso un semplice clic. In risposta, Argo UML presenta una serie di icone intorno al riquadro della classe, che è possibile trascinare verso un'altra classe per creare una relazione. Per instaurare un legame di ereditarietà è ovviamente necessario selezionare le icone che rappresentano la punta di freccia utilizzata per rappresentare questa relazione.
L'altro modo possibile è di selezionare la stessa icona nella barra degli strumenti e poi trascinare una riga dalla classe figlio a quella padre.

ASSOCIAZIONI

Ma l'ereditarietà non è la sola relazione che può esistere tra due classi. Molto più semplicemente, è possibile che una classe ne utilizzi un'altra come proprietà. In questo caso è possibile specificare il campo della classe come descritto nella puntata precedente. In alternativa, è possibile disegnare entrambe le classi nel diagramma e unirle con una riga continua. Quest'ultima possibilità è meno compatta rispetto alla prima, ma in alcuni casi può essere più chiara. Inoltre, è possibile arricchire la relazione scrivendo il nome della proprietà per cui è utilizzata quella classe direttamente sulla riga.
Questo tipo di rappresentazione grafica viene utilizzata quando la proprietà ha molteplicità 1. Questo significa che c'è una sola istanza della classe B per la classe A.

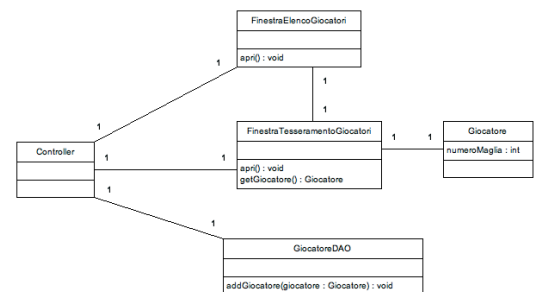
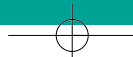



Fig. 4: Relazione tra due classi.

In **Figura 4** è presente un esempio che mostra le classi che potrebbero essere coinvolte nella funzionalità di gestione dell'elenco dei giocatori. Si trovano classi che rappresentano finestre applicative, entità di dominio, classi di controllo e accesso ai dati. Si noti che l'indicazione "1" agli estremi della relazione è totalmente opzionale.
Quando si esprime un attributo come relazione non è necessario descriverlo all'interno della classe. Le

due notazioni sono quindi esclusive.

AGGREGAZIONE E COMPOSIZIONE

L'associazione semplice va bene quando c'è una sola istanza della classe collegata B per quella principale A. Spesso però è necessario fare uso di strutture dati più complesse, che coinvolgano per esempio liste, mappe o vettori. Per esempio, ci possono essere diverse Partite per uno stesso Campionato, oppure fino a 11 Giocatori per ciascuna Squadra.
Per rappresentare questo tipo di situazioni UML dispone dei concetti di aggregazione e composizione. Entrambi hanno a che fare con relazioni 1 a N, ma con due livelli di "forza" diversi. L'aggregazione è una relazione meno forte, dove la classe padre esiste anche se non ci sono classi collegate. Inoltre, eventuali classi collegate continuano a esistere anche in modo indipendente. Un esempio potrebbe essere il team di lavoro e le risorse ad esso allocate: una volta dismesso il team, le risorse continuano ad esistere!
La composizione è un tipo di aggregazione più forte, dove la classe padre non ha senso di esistere se non ci sono le classi figlie e queste ultime non possono esistere da sole. L'esempio tipico è la relazione tra Fattura e ProdottoAcquistato.
Aggregazione e composizione si rappresentano in modo molto simile, utilizzando una linea di associazione che termina con un rombo. Se questo è pieno si tratta di una composizione. Se è vuoto di una aggregazione. In **Figura 5** è presente un diagramma di classe che utilizza soprattutto la composizione e mostra la relazione tra gli elementi principali dell'applicazione. Come si può notare dalla figura, la classe Campionato è associata all'oggetto Partita. Si tratta di una composizione. Non può infatti esistere un campionato senza partite, non avrebbe senso! A onor del vero, è possibile che esistano partita svincolate da un campionato. Sono le amichevoli, ma non sono contemplate in questa versione del programma. Come si può notare osservando la linea di relazione, è indicata una molteplicità di 30. Le partite in un campionato dovranno quindi essere in quell'esatto numero.
Anche Squadra è in relazione di composizione con Campionato, in quanto ci devono essere più squadre in uno stesso campionato. Per indicare un numero generico si può scrivere 0..*. Questo significa che la molteplicità va da zero a infinito. Considerando che è difficile fare un campionato con meno di otto squadre, si sarebbe anche potuto decidere per una molteplicità 8..*.
Infine, per ciascuna Partita è presente un oggetto

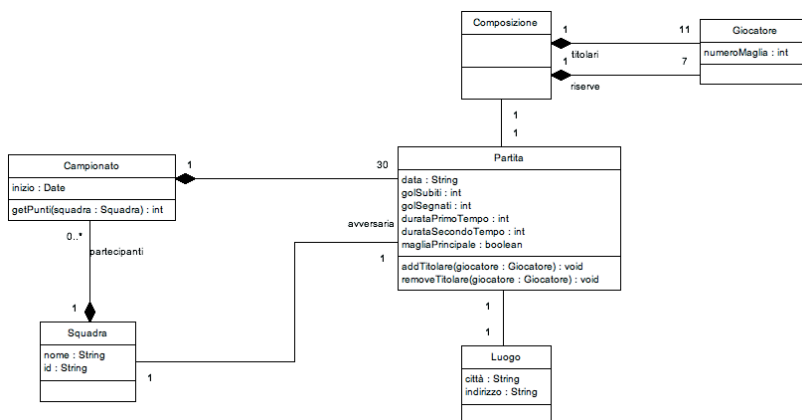



Fig. 5: Diagramma generale dell'applicazione Team2007

Composizione, che permette di descrivere la composizione della squadra per quella specifica partita. In un torneo Open il numero di riserve è di sette elementi, mentre il numero di giocatori è, come di consueto, di undici elementi. Queste molteplicità sono indicate sul diagramma esplicitamente.

DIPENDENZA

I legami di associazione, aggregazione e composizione non sono gli unici possibili in UML. Lo standard mette a disposizione anche un legame più lasco, che permette di indicare in generale che una classe ne utilizza un'altra. Si tratta del legame di "dipendenza". Questo tipo di relazione è rappresentato da una linea tratteggiata che termina con una punta di freccia aperta. È possibile descrivere la tipologia di dipendenza utilizzando una parola chiave racchiusa tra parentesi uncinate. UML definisce alcune parole chiave standard. Per indicare che una classe ne chiama un'altra si usa «call». Invece, per segnalare che una classe ne utilizza un'altra si indica «use». La parola chiave «create» si impiega invece per indicare che la classe principale crea istanze della classe secondaria.

Un libro decisamente economico ma che rappresenta un buon reference di tutti gli elementi di UML 2.0 è: Enrico Amedeo "UML Pocket", Editore Apogeo, ISBN: 978-88-503-2627-3.

CONCLUSIONE

In questa puntata è stato approfondito l'utilizzo di Argo UML ed è stata fornita qualche indicazione pratica per il suo utilizzo. Intanto abbiamo imparato qualche altra cosa sui diagrammi di classe, completando la trattazione dei suoi elementi fondamentali. Nella prossima puntata sarà il turno dei diagrammi di sequenza, indispensabili per modellare gli algoritmi e le interazioni tra le diverse classi dell'applicazione.

Massimiliano Bigatti

PROGRAMMARE GLI SMARTPHONE

AVETE MAI PENSATO DI PERSONALIZZARE PDA O CELLULARE CON APPLICAZIONI CREATE DA VOI? VB.NET CONSENTE DI FARLO PIUTTOSTO SEMPLICEMENTE MINIMIZZANDO I TEMPI DI SVILUPPO E FACILITANDO IL RILASCIO DI PROGRAMMI PER SMART DEVICE...

Al top dei desideri degli amanti di Pocket PC e Smartphone ci sono sicuramente il miglioramento delle performance del proprio device e l'immediata disponibilità del programma che soddisfi "magicamente" le necessità del momento. Se il primo desiderio di solito possiamo appagarlo semplicemente incrementando la memoria o, in casi estremi, acquistando un nuovo palmare o smartphone, il secondo è di più difficile soluzione. Anche ammettendo di avere un budget illimitato, non tutte le applicazioni commerciali possono fare al caso nostro e anche se un programma sembra essere quello giusto, ad un esame più attento, potrebbe risultare sovradimensionato o, comunque, non fornire al 100% le funzionalità che ci aspettiamo.
L'uovo di Colombo per Pocket PC e Smartphone equipaggiati con Windows Mobile è Microsoft Visual Basic .NET (da pronunciare "dot net"). La soluzione è, cioè, quella di creare con le nostre mani ciò che realmente ci serve in quel momento, con la possibilità di modificare il software successivamente e (scusate se è poco) senza pagare costose royalty a nessuno.
Microsoft Visual Basic.NET non é l'unico strumento che permette di programmare Pocket PC e Smartphone, ma è senz'altro il prodotto più semplice da utilizzare della famiglia Microsoft Visual Studio .NET.

REQUISITI

Conoscenze richieste
Conoscenze di base sulla programmazione

Software
Microsoft Visual Studio.NET 2005, Windows Mobile 5.0 Pocket PC SDK e Windows Mobile 5.0 Smartphone SDK

Impegno

Tempo di realizzazione

OBIETTIVI DEL CORSO

Il principale obiettivo è certamente ambizioso, perché questo corso intende insegnare in concreto e partendo dalle basi la programmazione di applicazioni mobili con Visual Basic.NET. Ciò significa che, passo dopo passo, tutti coloro che seguono il corso dovrebbero essere in grado di creare applicazioni per smart device. É necessaria una buona dose di pazienza e qualche conoscenza di base di Visual Basic.NET, per il resto tenteremo di guidarvi con esempi facili che perseguono obiettivi concreti.

UN PO' DI STORIA

Visual Basic.NET affonda le radici addirittura nel settembre 1963 quando, negli Stati Uniti presso il Dartmouth College, venne dato inizio ai lavori per la creazione del Beginners All-purpose Symbolic Instruction Code (che significa più o meno "linguaggio simbolico per principianti adatto per usi generali") sotto la direzione dei professori John G. Kennedy e Thomas E. Kurtz. Si trattava della prima versione del BASIC, il cui varo effettivo avvenne il

Fig. 1: Un'applicazione per smart device che esegue un dizionario all'interno di un emulatore per Pocket PC

1 maggio 1964, alla conclusione dei lavori portati avanti per lo più da studenti universitari. Una delle caratteristiche basilari del BASIC sin dall'inizio era la sua facilità di utilizzo; lo sforzo dei pionieri del Datmouth College fu proprio rivolto alla produzione di un linguaggio ad alto livello, che privilegiasse la chiarezza a discapito delle prestazioni.
Successivamente, verso la metà degli anni '70, furono Bill Gates e Paul Allen, i fondatori della Microsoft, a sviluppare una versione riveduta e corretta del BASIC per il primo Personal Computer commercializzato, l'Altair. Si trattava del GW-BASIC, il BASIC interpretato, che, da quel momento in poi, venne fornito con tutte le versioni del sistema operativo DOS.
Il varo della nuova veste grafica si ebbe alla fine degli anni '80, quando sul mercato comparve Microsoft Visual Basic 1.0. Il nuovo prodotto destò subito una buona impressione tra gli addetti ai lavori, ma la consacrazione definitiva è dell'inizio degli anni '90 con la distribuzione di Visual Basic 3.0. Il resto è storia dei nostri giorni: dopo una versione interlocutoria, la 4.0, Visual Basic viene incorporato nelle suite Visual Studio (la prima all'inizio del 1997) e da quel momento in poi ne farà sempre parte, fino ad arrivare a Visual Studio.NET.

REQUISITI NECESSARI PER LAVORARE

L'hardware di cui abbiamo bisogno consiste in un PC possibilmente di ultima generazione, per garantirci prestazioni accettabili. In effetti un qualsiasi PentiumIV con almeno 512KB di memoria RAM va più che bene. Dobbiamo, poi, procurarci un device mobile con cavo di connessione USB per poterci collegare al PC e testare come funzionano le applicazioni che abbiamo sviluppato. Il sistema operativo del PC dovrà essere Microsoft Windows XP o Vista (ma in quest'ultimo caso le esigenze hardware aumentano) e Microsoft Windows Mobile 5 per il device mobile.
Ci serve ovviamente anche Microsoft Visual Studio .NET. La versione attualmente commercializzata è la 2005. Se si decide di acquistare il prodotto è consigliabile richiedere almeno la distribuzione Professional che permette di creare applicazioni per Windows, eseguibili direttamente all'interno del sistema operativo; Office, da eseguire all'interno di prodotti come Word ed Excel; Smart device, ossia Pocket PC e Smartphone con sistema operativo Windows Mobile 2003 o la release Windows Mobile 5.0. E', inoltre, necessario installare Microsoft ActiveSync liberamente scaricabile dal sito Microsoft (attualmente l'ultima versione è la 4.5) che consente la sincronizzazione della periferica mobile con il PC.

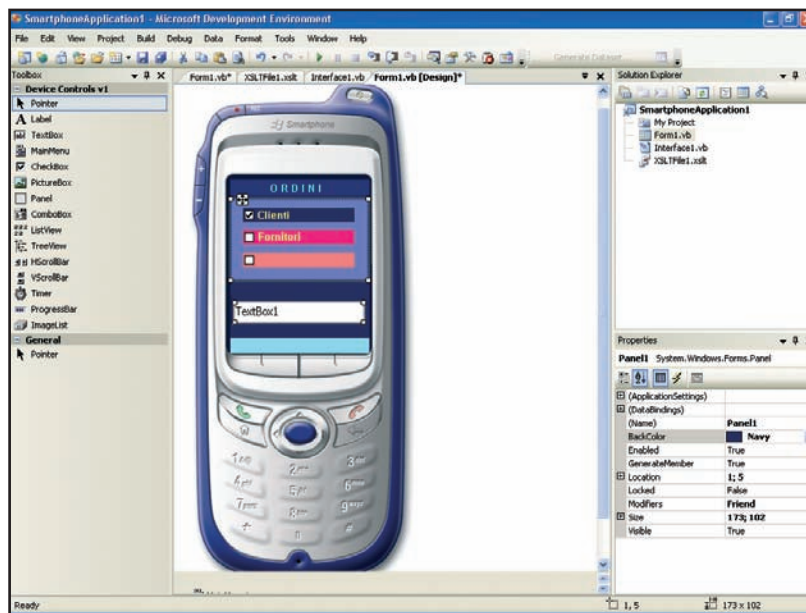



Fig. 2: Un'applicazione per Smartphone a design time

Infine, sempre dal sito Microsoft, dobbiamo scaricare una risorsa aggiuntiva necessaria per creare applicazioni secondo lo standard Pocket PC e Smartphone 2005: Microsoft Pocket PC 2005 SDK.msi e Windows Mobile 5.0 Smartphone SDK.msi.

INSTALLARE VB.NET E L'SDK

L'installazione dell'ambiente di sviluppo è semplice e completamente guidata. Si tratta d'inserire il CD contenente Microsoft Visual Studio .NET e, una volta visualizzata la schermata con le varie opzioni software, di scegliere l'installazione completa di Visual Basic (gli altri linguaggi, almeno per ora, non ci servono). Una volta ultimata l'installazione di VB.NET procediamo a quella del SDK (Software Development Kit, il kit di sviluppo software per Pocket PC 2005). Si tratta di fare doppio click sul file scaricato in precedenza Microsoft Pocket PC 2005 SDK.msi, rispondendo, poi, OK all'osservazione che Visual C++ 4.0 non è presente sul PC e scegliendo l'installazione completa. Procedete, poi, all'installazione anche del SDK per smartphone e cioè di Windows Mobile 5.0 Smartphone SDK.msi. Fatto questo siamo quasi pronti per iniziare.

NOTA

DIFFUSIONE DI VB.NET

Un impulso decisivo alla diffusione di Visual Basic.NET la sta dando proprio la crescita esplosiva delle vendite dei dispositivi mobili. Pocket PC e Smartphone diventano sempre più potenti sia in termini di potenza elaborativa che di capacità di visualizzazione e, di conseguenza, rendono sempre più interessanti gli ambienti di programmazione per smart device.

088-094 29-06-2007 16:55 Pagina 90

APPLICAZIONI LOCALI E NON

Le applicazioni per smart device possono avere caratteristiche diverse, ma essenzialmente si raggruppano in tre categorie che esaminiamo di seguito:

1. Connesse, ovvero quelle che hanno sempre bisogno di un computer con cui dialogare. Un esempio può essere un'applicazione anagrafica che contiene, magari, i dati di tutti i nostri clienti, dove l'interfaccia per interrogare, inserire e modificare le informazioni risiede sul dispositivo mobile mentre il database che contiene i dati anagrafici si trova su un computer server. In questo caso, se il server che ospita il database e/o la connessione di rete sono indisponibili, l'applicazione non può essere utilizzata ed è virtualmente inutile.

2. Non sempre connesse, ossia quelle che hanno solo saltuariamente bisogno di un computer raggiungibile via rete. Ad esempio un'applicazione dove l'intero database risiede su un computer server, mentre interfaccia di gestione e le tabelle, in cui vengono replicate solo determinate informazioni a richiesta degli utenti, si trovano sul device mobile. In questo caso si ha la possibilità di lavorare anche off-line, anche se con evidenti limitazioni.

3. Applicazioni stand-alone. Si tratta di applicazioni come un visualizzatore d'immagini, un software di videoscrittura o altro ancora, che non necessitano assolutamente di connessioni di rete o di computer remoti. Esse vivono di vita propria: tutto ciò di cui hanno bisogno é già stato installato nel Pocket PC o nello smartphone.

L'ICONA DEL PROGETTO

Per incorporare un icona nel progetto dovete crearla (Add-New Item e fate clic su Icon File) oppure fare riferimento ad una giá esistente (Add-Existing Item e trovate il percorso dell'icona da incorporare). Successivamente in Project-<nome progetto>Properties fate clic sulla scheda Application ed includete un'icona nel campo Icon.

L'AMBIENTE DI LAVORO

Non appena eseguite Visual Basic.NET per la prima volta vi viene richiesto di definire il linguaggio di default, ossia quello che utilizzerete più di frequente, VB.NET appunto.
La Start Page che compare in breve tempo fornisce indicazioni di carattere generale, cercando anche di collegarsi su Internet per visualizzare notizie provenienti dal mondo degli sviluppatori Visual Studio.
Attivando il menu File-New Project è possibile scegliere il tipo di progetto da aprire. Selezionate, ad esempio, Smart Device, Windows Mobile 5.0 Smartphone e Device Application per visualizzare la Form1, ossia l'oggetto grafico su cui creare il front-end della vostra applicazione. Se, poi, scegliete View-Solution Explorer potete visualizzare la finestra del progetto che contiene tutti gli oggetti che ne fanno parte. View-Properties Window, invece, visualizza la finestra delle proprietà dell'oggetto attivo. Ciascuna proprietà consente di personalizzare velocemente alcune caratteristiche, come il colore di sfondo, il tipo di font o il nome che identifica univocamente quell'oggetto.
Con View-Toolbox, infine, richiamate la casella degli strumenti che possono essere riportati sulla form. Gli oggetti sono suddivisi in varie sezioni e sono quasi tutti visualizzabili selezionando semplicemente All Windows Form sulla Toolbox stessa. Le loro particolarità sono tali e tante da renderne impossibile un esame generale. E', comunque, importante raggrupparli in due grandi categorie: quelli che sono immediatamente usufruibili e visi-

Fig. 4: Prima di aprire un nuovo progetto possiamo scegliere tra diverse opzioni.

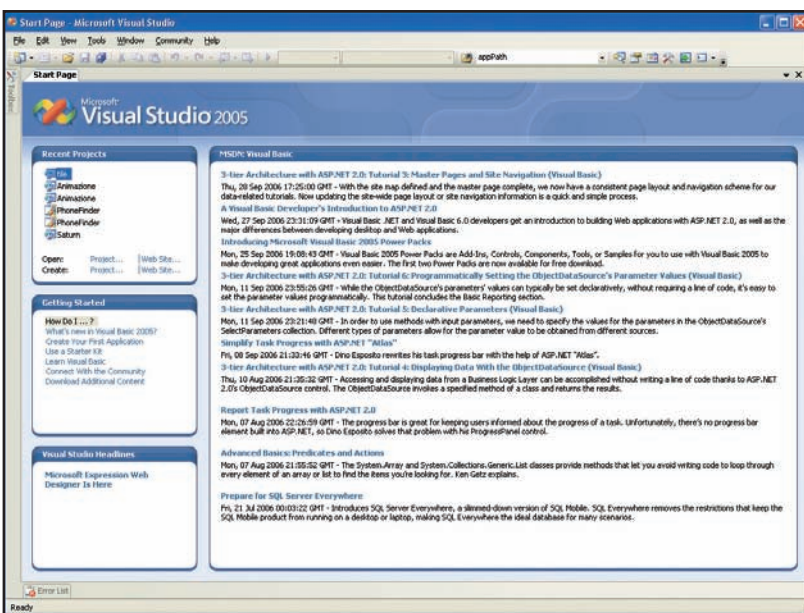



Fig. 3: L'ambiente di sviluppo di Visual Studio.NET, appena avviato, fornisce una carrellata di link a documenti utili per la programmazione.

bili sulla form e quelli che, pur mettendo a disposizione dei servizi, devono essere accuratamente personalizzati prima di poterli utilizzare. Un esempio dei primi sono il Button e la Label, dei secondi la ImageList e il Timer.

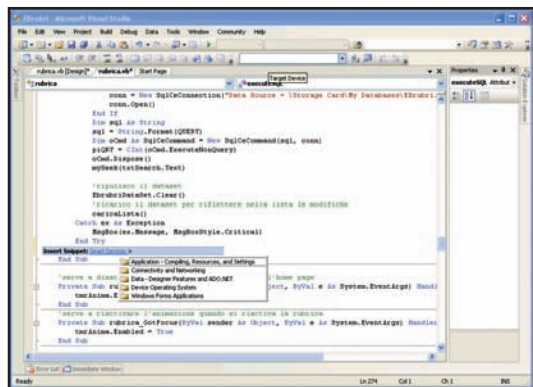



Fig. 5: Gli snippet di default per smart device comprendono ben 5 categorie di algoritmi differenti

UTILIZZARE GLI SNIPPET

La presenza di Snippet, ossia di porzioni di codice richiamabili dal text editor in qualsiasi momento, rende ancora più rapido lo sviluppo (RAD = Rapid Application Development). È sufficiente, infatti, fare clic destro col mouse e

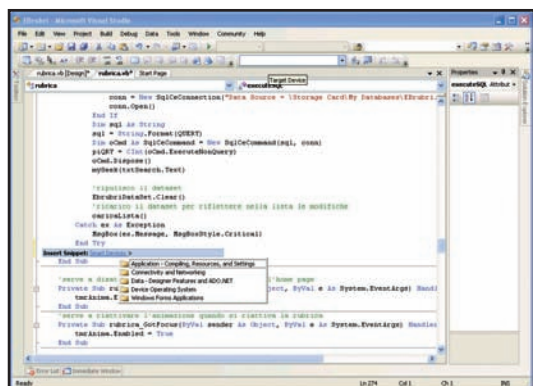



Fig. 6: Nella figura si evidenzia il percorso per risalire agli snippet per smart device

selezionare Insert Snippet per accedere ad una libreria di componenti che permettono di risparmiare tempo durante la fase di sviluppo. Trattandosi di porzioni di codice riusabile sono sempre disponibili e sono utili per gli scopi più disparati. Ad esempio per impostare algoritmi matematici, come il calcolo di una rata, oppure funzioni trigonometriche. Sono, inoltre, personalizzabili. Il codice è presente nel percorso \Programmi\Microsoft Visual Studio 8\Vb\Snippets\1033 ed è in formato XML. Possono essere richiamati dal disco locale o da Internet, cercandoli, ad esempio, nella libreria MSDN.

Infine viene fornito un apposito editor per creare o modificare gli snippet, poiché noi stessi possiamo crearne di nuovi, rendendoli disponibili da quel momento in poi in fase di sviluppo di nuove applicazioni.

In particolare per utilizzare gli snippet appositamente creati per i device mobili é necessario fare clic destro sull'editor del codice e scegliere Insert Snippet e, poi, Smart Devices. Successivamente è possibile selezionare una qualsiasi delle opzioni che risultano dalla lista di riepilogo. Se scegliete Windows Forms Applications e, poi, Drawing avrete a disposizione ben dieci algoritmi già "preconfezionati" e pronti ad essere utilizzati; ad esempio "Create Image with Transparency" consente di creare in pochi attimi immagini con effetto trasparente, utili per rendere piú accattivante l'interfaccia della vostra applicazione.

IL PRIMO PROGRAMMA PER PPC

Per creare la prima applicazione dovete avvia-

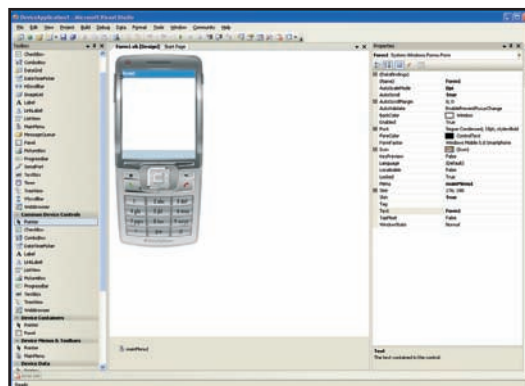



Fig. 7: La form di un progetto per Pocket PC.

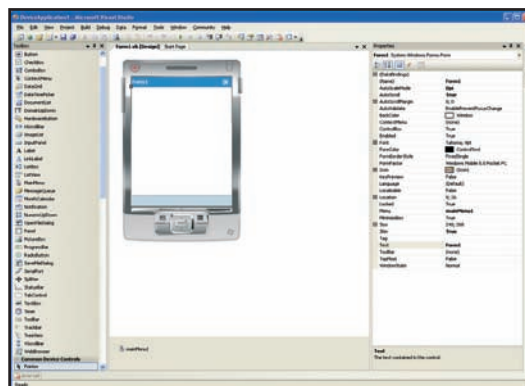



Fig. 8: La form di un progetto per Smartphone.

NOTA

LA FUNZIONE MSGBOX()

MsgBox() visualizza un messaggio in una finestra di dialogo e attende che l'utente faccia clic su un pulsante. E' una funzione standard di Visual Basic.NET ed accetta piú parametri per fare il display del testo, del titolo messaggio, dei pulsanti che compaiono al suo interno e anche dell'icona visualizzata.

088-094 29-06-2007 16:55 Pagina 92

re Visual Basic .NET, fare clic sul menu File-New Project e fare clic su Smart Device nella sezione Project Types e, poi, ancora su Windows Mobile 5.0 Pocket PC. Nella schermata che segue fate semplicemente clic su OK. In pochi attimi VB.NET visualizza un oggetto rettangolare, chiamato form, che non è altro che lo "sfondo grafico" del Pocket PC, su cui noi a poco a poco aggiungeremo degli oggetti e creeremo l'applicazione da trasferire, poi, sul nostro Pocket PC.
Di solito tutti i programmi per Pocket PC sono

Fig. 9: LImpostando a False la proprietà Skin della form viene visualizzato un oggetto form che non visualizza l'emulatore del PPC o dello Smartphone.

ISTRUZIONE WITH...END WITH

Consente di portare a fattor comune un oggetto, eseguendo una serie di istruzioni su di esso senza doverlo ripetere all'interno del blocco With. La sua sintassi è la seguente:
With oggetto [istruzioni]End With

costituiti almeno da due elementi: la parte (interfaccia) grafica, cioè ciò che si vede sul monitor e il cosiddetto motore, cioè le istruzioni software che danno vita all'applicazione.
Come prima cosa visualizzate la Toolbox e fate doppio clic sull'oggetto Button, per riportarlo all'interno della Form1.
Questo oggetto è un pulsante di comando (simile agli onnipresenti pulsanti del mondo Windows) ed, in fase di esecuzione, può essere premuto in maniera tale da scatenare le azioni che sono state programmate al suo interno. Si tratta di un classico esempio di oggetto programmabile, ma che ha dei comportamenti predefiniti che non possono essere modificati (si pensi alla particolarità di simulare un movimento ogni volta che viene cliccato).
Come tutti gli oggetti di VB.NET, form compresa, Button include un certo numero di proprietà predefinite che possono essere modificate direttamente sulla finestra delle proprietà. Aprite, dunque, tale finestra facendo un solo clic sinistro sul pulsante stesso e premendo, poi, F4. Modificate *Location* inserendo i valori 80,200 al posto di 0,0 e Text sostituendo Button1 con TEST.
Inserite ora nella form anche una TextBox con il solito doppio clic sulla Toolbox. Questo oggetto in fase di esecuzione permette di digitare del testo al suo interno. Modificate ora alcune proprietà (F4) della TextBox: in BackColor inseriamo SkyBlue, in ForeColor digitiamo Blue, in Location inseriamo i valori 40,170 e cancellate il valore (TextBox1) di Text.
Quando si avvia un nuovo progetto viene inserito automaticamente l'oggetto mainMenu.
Poiché per ora non ci serve selezionate mainMenu1 nella barra sotto la form e cancellatelo.
Infine inserite un po' di codice nel solo pulsante. Per fare questo fate doppio clic su Button1 in maniera tale da attivare lo spazio destinato all'evento clic (Private Sub Button1_Click) e inserite l'algoritmo che segue:

```
If TextBox1.Text <> "" Then
    TextBox1.BackColor =
                System.Drawing.Color.Green
    TextBox1.ForeColor =
                System.Drawing.Color.Yellow
    MsgBox(TextBox1.Text)
    TextBox1.Text = ""
    TextBox1.BackColor =
                System.Drawing.Color.SkyBlue
    TextBox1.ForeColor =
                System.Drawing.Color.Blue
End If
```

D'ora in poi tutte le volte che in fase di esecuzione si fa clic sul pulsante TEST viene attivato l'evento *Button1_Click* che determina l'esecuzione dell'algoritmo appena descritto. La spiegazione del codice in esso contenuto è semplice. Si verifica subito con un'istruzione IF se il testo è cambiato, cioè se *TextBox1.Text* è diverso (<>) da spazio vuoto (""). Solo in questo caso vengono eseguite determinate istruzioni:

- modifica del colore di sfondo della TextBox, che diventa verde (*TextBox1.BackColor = System.Drawing.Color.Green*)

- modifica del colore in primo piano della TextBox, che diventa giallo (*TextBox1.ForeColor =*

088-094 29-06-2007 16:55 Pagina 93

System.Drawing.Color.Yellow)

- visualizzazione di un messaggio che contiene proprio quanto è stato digitato e cioè *TextBox1.Text* (*MsgBox(TextBox1.Text)*)

- una volta scomparso il messaggio (clic su OK), viene ripulita la TextBox (*TextBox1.Text* =“”) e s’impostano di nuovo i colori come in origine (*TextBox1.BackColor= System.Drawing.Color.SkyBlueeTextBox1. ForeColor=System.Drawing.Color.Blue*).

Ovviamente se non digitate mai nulla sulla TextBox non accade niente, perché la IF non rileva cambiamenti e ci manda dopo l’End If, terminando il programma. A questo proposito se volete evidenziare che l’utente deve prendere un’azione potete modificare il codice precedente come segue:

```
If TextBox1.Text <> "" Then
    TextBox1.BackColor =
                    System.Drawing.Color.Green
    TextBox1.ForeColor =
                    System.Drawing.Color.Yellow
    MsgBox(TextBox1.Text)
    TextBox1.Text = ""
    TextBox1.BackColor =
                    System.Drawing.Color.SkyBlue
    TextBox1.ForeColor =
                    System.Drawing.Color.Blue
Else
    TextBox1.BackColor =
                    System.Drawing.Color.Red
    MsgBox("Digitare almeno un carattere nella
                    TextBox !!")
    TextBox1.BackColor =
                    System.Drawing.Color.SkyBlue
End If
```

È stato aggiunto un ramo Else (= altrimenti) nel quale viene colorata di rosso la TextBox e visualizzato un messaggio che richiede la digitazione di qualche carattere. In ogni caso dopo la MsgBox la TextBox viene colorata di nuovo di azzurro.

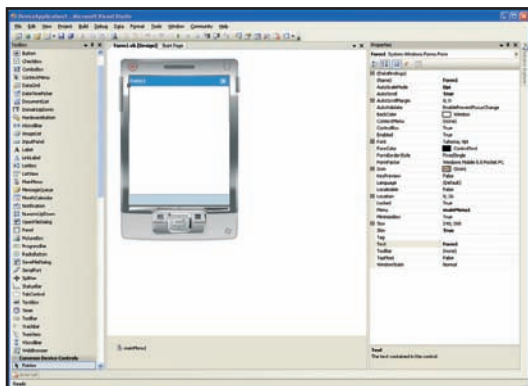



Fig. 10: L’editor del codice facilita l’immissione degli algoritmi grazie ai suoi meccanismi automatici.

PROGRAMMAZIONE PER EVENTI

VB.NET, oltre a fornire strumenti automatici per creare l’interfaccia grafica, rende disponibile per ciascun oggetto eventi che possono essere programmati a seconda delle necessità. Ad esempio se si sceglie (*Form1 Events*) nell’editor del codice, a destra si possono selezionare gli eventi disponibili per ciascuna form. Se, dunque, volete eseguire un algoritmo che colora la form di celeste quando viene caricata in memoria, non dovete fare altro che inserire nell’evento *Load* l’istruzione *Me.BackColor = System.Drawing.Color.Cyan*. Appena viene eseguita l’applicazione lo sfondo della form diventerà celeste, perché avete inzializzato la proprietà *Backcolor* di Me (che rappresenta la form corrente) con il colore Cyan (ossia celeste).

```
Private Sub Form1_Load(...) Handles
                            MyBase.Load
    Me.BackColor = System.Drawing.Color.Cyan
End Sub
```

Così se voleste visualizzare un certo messaggio tutte le volte che viene fatto clic sulla form dovete inserire il codice nell’evento Click della form stessa come segue.

```
Private Sub Form1_Click(...) Handles Me.Click

    MsgBox("Applicazione sviluppata con
                    VB.NET da ENRICO")
End Sub
```

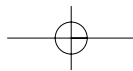
Ogni evento ha la sua utilità e tutti i controlli, come pulsanti, etichette, immagini sono dotati di eventi. Tuttavia solo una minima parte di essi vengono normalmente utilizzati in un’applicazione: in particolare vengono programmati solo quegli eventi necessari allo sviluppo della logica dell’applicazione.

UNA NOTA DI STILE

La programmazione oltre che essere una scienza è anche un’arte, ecco perché di seguito si propone un miglioramento stilistico del programma sino ad ora sviluppato. Si noti che

NOTA

RISORSE SUL WEB

Potete reperire utili informazioni, consigli e suggerimenti su Visual Basic.NET visitando i seguenti siti:
www.visualbasic.it
www.visual-basic.it
www.visual-basic.com
www.ugidotnet.org
www.visualbasic.net
www.vbwm.com
www.codeguru.com
msdn.microsoft.com/library (sezione Development Tools and Languages)
msdn2.microsoft.com/en-us/vbasic

le prestazioni non variano e che da un punto di vista logico il programma è rimasto perfettamente uguale a prima, tuttavia il codice risulta molto più compatto e più facilmente leggibile.

```
Imports System.Drawing.Color
...
    With TextBox1
        If .Text <> "" Then
            .BackColor = Green
            .ForeColor = Yellow
            MsgBox(.Text)
            .Text = ""
            .BackColor = SkyBlue
            .ForeColor = Blue
        Else
            .BackColor = Red
            MsgBox("Digitare almeno un carattere
                                nella TextBox !!")
            .BackColor = SkyBlue
        End If
    End With
```

Fig. 11: L'emulatore in esecuzione visualizza due differenti comportamenti a seconda del ramo della If che viene seguito

Innanzitutto abbiamo aggiunto l'istruzione *With...End With* prima dell'*If*. Questo ci serve per evitare di ripetere n volte *TextBox1*, che viene, quindi, evidenziata solo nella *With*.
Successivamente abbiamo indicato solo il nome del colore (Green, Yellow, ecc.) grazie all'istruzione *Imports*, che consente di portare a fattor comune la *System.Drawing.Color*. Si ricordi infine che la Imports deve sempre essere riportata come prima dichiarazione in una form.

ESECUZIONE DELL'APPLICAZIONE

Il ciclo di vita di un programma prevede sempre almeno due fasi: 1. quella di progettazione o sviluppo o disegno (design time); 2. quella di esecuzione (run time).
La fase di progettazione l'abbiamo appena vista: creazione della form con i propri oggetti, definizione delle proprietà e inserimento del codice.
La fase di esecuzione è quella che ci consente di vedere come l'applicazione funzionerà sul Pocket PC o sullo Smartphone.
Se facciamo clic sul menu Tools-Options (opzione Device Tools e, poi, Devices) possiamo subito visualizzare le opzioni di esecuzione che abbiamo a disposizione. Le possibilità che c'interessano, nel caso di un programma per Pocket PC, sono le due che seguono:
1. Windows Mobile 5 Pocket PC Emulator. In questo caso l'applicazione viene visualizzata all'interno dell'emulatore.
2. Windows Mobile 5 Pocket PC Device. In questo caso l'applicazione viene distribuita al Pocket PC connesso al PC e visualizzata al suo interno. Per attivare la fase di esecuzione in entrambi i casi bisogna prima fare clic sull'icona con la freccetta oppure scegliere il menu Debug-Start Debugging. Dopo avere selezionato una delle due opzioni precedenti bisogna fare clic sul pulsante Deploy. Per adesso ci fermiamo qui. Dovreste essere già in grado di sviluppare la vostra prima semplice applicazione. Nei prossimi numeri ci soffermeremo su altri elementi essenzial per programmare facilmente applicazioni dedicate al mondo del Mobile. Come abbiamo visto gli strumenti non sono complessi, non ci rimane che fare un po' di pratica

Enrico Bottari

L'STRUZIONE IF...THEN...ELSE

Essa consente l'esecuzione condizionale di un gruppo d'istruzioni, in base al valore di un'espressione. La sua sintassi è la seguente:

```
If condizione Then [istruzioni]
Else       [istruzioni])
End If
```

Quando si utilizza una If bisogna inserire obbligatoriamente le due parole riservate Then ed End If. In questo caso fortunatamente l'editor di VB.NET ci aiuta perché basta digitare una If con la condizione e dare Invio per vedere in automatico l'inserimento delle due parole riservate. La Else è facoltativa e puó essere omessa se non è necessario specificare un'alternativa.

USIAMO JSF CON I MANAGED BEAN

IMPARIAMO COME SI GESTISCONO I DATI TRAMITE LE FORM, COME È POSSIBILE VALIDARLI E COME STAMPARE A SCHERMO LE INFORMAZIONI FORNITE IN INPUT. VEDREMO COME IL CICLO DI VITA DI UN'APPLICAZIONE JSF È DECISAMENTE DIVERSO DA QUELLO CLASSICO

In questo articolo vedremo una prima applicazione sviluppata con JSF, dove andremo ad utilizzare i managed bean, oggetti che ci permettono di modellare i dati inviati dall'utente e la loro logica applicativa.

CICLO DI VITA

Prima di iniziare a vedere in maniera più approfondita un'applicazione JSF, dobbiamo capire come viene gestito il ciclo di vita della nostra applicazione se utilizziamo questo framework. Se siamo abituati a sviluppare utilizzando i classici strumenti che mette a disposizione Java in ambito EE, ovvero JSP e Servlet, saremo abituati ad un ciclo di vita abbastanza semplice. Come primo punto abbiamo la richiesta da parte di un client di un certo URL. La nostra Servlet/JSP che gestisce questa HttpRequest, prepara un HttpResponse in base alla logica della nostra applicazione e successivamente risponde al client.

Questa è una visione abbastanza semplicistica del ciclo di vita nella applicazioni Java EE, ma d'altronde è l'ossatura su cui si basano tutte le applicazioni client/server, ovvero request e response. La nostra applicazione viene calata in questo ciclo di vita e trova posto all'interno della gestione delle richieste che vengono effettuate dal client. Chiunque di voi abbia realizzato un'applicazione web saprà sicuramente che, oltre a questa semplice visione dell'applicazione, entrano in giochi tantissimi elementi (gestione dati, validazione, invocazione di metodi etc. etc.). Il ciclo di vita di JSF non fa altro che definire le fasi caratteristiche di un'applicazione web, entrando nel dettaglio di ognuna. In questo modo possiamo modellare la nostra applicazione su queste fasi del ciclo di vita, sviluppando per ogni punto il codice di cui abbiamo bisogno per completare un determinato use case.

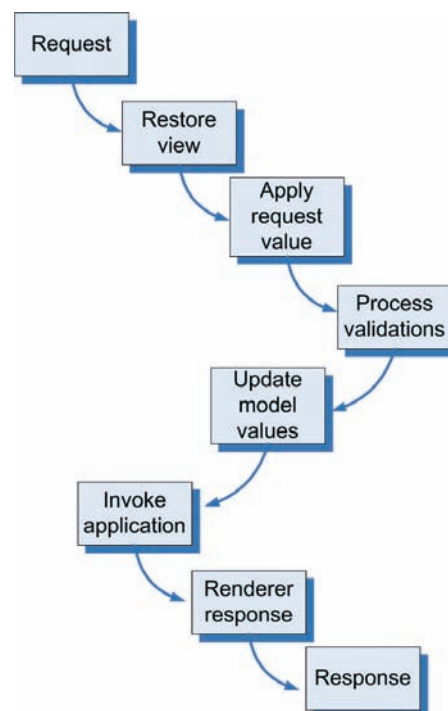
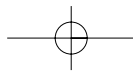



Fig. 1: Ciclo di vita standard

Fig. 2: Ciclo di vita di JSF

095-101 29-06-2007 16:52 Pagina 96

Vediamo quindi, punto per punto, cosa succede nel classico ciclo di vita della nostra applicazione JSF. Il primo stato è "Restore View", dove il framework JSF cerca di capire quale sia la view presente sul client, per poter poi gestire i suoi componenti. Questa informazione è presente all'interno della request che invia il client, attraverso un ID della vista. Una volta che viene trovato questo ID, la view viene recuperata. Chiaramente se l'utente visita per la prima volta quella pagina, non avranno nella sua request nessun valore che specifica la view, quindi il framework dovrà provvedere ad istanziare una nuova vista e a registrarla nel contesto. Le successive due fasi possono essere considerate insieme: "Apply request values" e "Process validation". In queste fasi vengono recuperati, convertiti e valicati i valori relativi ai dei componenti utilizzati nella nostra applicazione. La validazione dei valori può essere gestita tramite le regole standard definite da JSF (ovvero si specifica che un certo attributo di un certo componente deve essere un intero) oppure tramite dei metodi di validazione che lo sviluppatore deve implementare (mi vengono passate due date per la prenotazione di un viaggio e deve essere sicuro che rientrino in un certo range). Passiamo quindi alla fase "Update model values", dove vengono salvati i parametri inviati dall'utente nei rispettivi componenti (sicuri che l'informazione che andiamo a settare/modificare è valida). La fase successiva, "Invoke application", è quella dove possiamo inserire dei metodi di business in base all'evento di request che è stato generato o in base alle informazioni che sono state inviate dal client. Infine c'è la fase di "Render response", dove vengono visualizzati tutti i componenti nella response che deve essere inviata al client. All'interno della nostra applicazione, non dovremo sicuramente utilizzare tutti le fasi descritte per ogni singola interazione, quindi semplicemente se in base ad una certa request vogliamo soltanto invocare un metodo di business, salteremo le fasi precedenti e andremo direttamente alla costruzione della response. Vediamo ora un primo esempio di applicazione JSF.

NOTA

JSF, vista la sua natura orientata all'evento, può essere associato in maniera molto semplice allo sviluppo di applicazioni AJAX
http://www.theserverside.com/news/thread.tss?thread_id=40010
http://java.sys-con.com/read/171490.htm
https://blueprints.dev.java.net/ajaxcomponents.html

INSTALLAZIONE

Per incominciare ad utilizzare JSF dobbiamo prima di tutto configurarlo nel nostro ambiente di sviluppo. SUN permette di scaricare dal proprio sito l'implementazione 1.2 di JSF (http://java.sun.com/javaee/javaserverfaces/download.html) che possiamo utilizzare tranquillamente nei nostri progetti. Esiste anche un'alternativa all'implementazione della SUN, un progetto opensource di Apache, MyFaces (http://myfaces.apache.org/). Questo progetto mira a diventare il punto di riferimento per gli sviluppatori JSF, fornendo oltre agli strumenti definiti nelle specifiche anche altre componenti interessanti per lo sviluppo. Per iniziare a vedere JSF vi consiglio di cominciare ad utilizzare l'implementazione della SUN, poi in un secondo momento è possibile vedere qual è l'implementazione che più ci aggrada. Una volta che abbiamo scaricato l'implementazione JSF (https://javaserverfaces.dev.java.net/), possiamo trattarla come una qualsiasi libreria che andiamo ad aggiungere alla nostra Web Application, quindi nella cartella WEB-INF/lib del nostro progetto o nella cartella shared/lib di Tomcat. Chiaramente possiamo anche evitare di scaricare JSF e copiarlo in queste cartelle se l'IDE che utilizziamo ci permette di gestire questo framework. NetBeans (che verrà utilizzato in questo articolo) e Eclipse hanno dei plugin che possono essere molto utili per lo sviluppo con JSF.

Fig. 2: NetBeans comprende al suo interno la distribuzione SUN di JSF

APPLICAZIONE

Con l'applicazione che ora realizzeremo riusciremo a capire meglio alcune cose del framework JSF, perché come sempre è più facile fare una cosa che cercare di capire come si fa. L'applicazione che svilupperemo ora ci permette di definire la nostra biblioteca, inserendo e visualizzando tutte le informazioni relative ai libri. In questo modo vedremo come poter definire dei bean che gestiscono i dati inviati dall'utente tramite form, come validarli e come invocare delle operazioni all'interno della logica della nostra applicazione.

HOMEPAGE

Nell'homepage della nostra semplice applicazione vogliamo visualizzare tutte le diverse azioni che è possibile fare. Per fare ciò dobbiamo creare una semplice JSP con un paio di link, ognuno dei quali è collegato ad una diversa azione

```
<%@ page contentType="text/html"%>
<%@ taglib uri="http://java.sun.com/jsf/core"
                                        prefix="f"%>
<%@ taglib uri="http://java.sun.com/jsf/html"
                                        prefix="h"%>
<f:view>
  <html>
    <head>
      <title>La mia Biblioteca</title>
    </head>
    <body>
      <h:form>
        <h2>Menu</h2>
```

095-101 29-06-2007 16:52 Pagina 97


```
        <h:commandLink action="insert">
          <h:outputText value="Inserimento libro"/>
        </h:commandLink>
        <h:commandLink action="list">
          <h:outputText value="Lista catalogo"/>
        </h:commandLink>
      </h:form>
    </body>
  </html>
</f:view>
```

Abbiamo definito due diversi link, con un determinato valore del testo visualizzato e dell'azione che provocano. Cliccando sul primo link invieremo alla nostra applicazione JSF l'informazione "dalla pagina biblioteca.jsp è stato ricevuto l'evento insert". Sulla base di questa informazione l'applicazione JSF consulta la propria configurazione e decide su quale pagina inviarci. In questo caso vogliamo visualizzare la pagina insert.jsp, dove sarà presente il form d'inserimento. Per fare ciò dobbiamo inserire la seguente configurazione all'interno del file faces-config.xml, che si trova nella sottodirectory WEB-INF della nostra applicazione JSF

```
<navigation-rule>
      <from-view-id>/biblioteca.jsp</from-view-id>
      <navigation-case>
            <from-outcome>insert</from-outcome>
            <to-view-id>/insert.jsp</to-view-id>
      </navigation-case>
      <navigation-case>
            <from-outcome>list</from-outcome>
            <to-view-id>/list.jsp</to-view-id>
      </navigation-case>
</navigation-rule>
```

DEFINIZIONE DEL BEAN

Prima di creare la pagina per l'inserimento dobbiamo realizzare il bean nel quale andremo a salvare tutte le informazioni relative al singolo libro. Questo bean è una semplice classe Java, con una serie di variabili e i relativi metodi get e set.

```
package it.ioprogrammo.jsf;

import java.util.Date;
import javax.faces.application.FacesMessage;
```

```
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.validator.ValidatorException;

public class Libro {

    public String titolo;
    public String autori;
    public String editore;
    public Date pubblicazione;
    public String isbn;
    public double prezzo;

    public Libro() {
    }

    public String getTitolo() {
        return titolo;
    }

    public void setTitolo(String titolo) {
        this.titolo = titolo;
    }

    public String getAutori() {
        return autori;
    }

    public void setAutori(String autori) {
        this.autori = autori;
    }

    public String getEditore() {
        return editore;
    }

    public void setEditore(String editore) {
        this.editore = editore;
    }

    public Date getPubblicazione() {
        return pubblicazione;
    }

    public void setPubblicazione(Date pubblicazione) {
        this.pubblicazione = pubblicazione;
    }

    public String getIsbn() {
        return isbn;
    }

    public void setIsbn(String isbn) {
        this.isbn = isbn;
    }

    public double getPrezzo() {
```

NOTA

JSF mette a disposizione diversi componenti ma è possibile comunque creare un proprio componente utilizzando le API del framework
http://today.java.net/pub/a/today/2004/07/16/jsfcustom.html
http://www.ibm.com/developerworks/java/library/j-jsf4/
http://www.artima.com/lejava/articles/javaone_2007_stephane_bastian.html

```
        return prezzo;
    }

    public void setPrezzo(double prezzo) {
        this.prezzo = prezzo;
    }

    public void validateISBN(FacesContext context,
                                UIComponent toValidate,
            Object value) throws ValidatorException {
        String isbnSubmit = (String) value;

        if (isbnSubmit.length()<9) {
            FacesMessage message = new
                        FacesMessage("Invalid ISBN");
            throw new ValidatorException(message);
        }
    }

    public String addLibro() {
        //Codice per la connessione ad un DB
        System.out.println("Libro aggiunto alla
                                        libreria");
        return "aggiunto";
    }
}
```

Come potete vedere ci sono due metodi in più rispetto al classico bean, validateISBN() e addLibro(). Il primo metodo serve per creare una nostra regola di validazione. Mentre gli altri valori verranno controllati con dei validatori standard di JSF, il campo ISBN verrà controllato da noi per poterlo valicare nella maniera che vogliamo. Questo serve soltanto per far capire come sia possibile gestire la validazione dei parametri che sono inseriti nelle form dal punto di vista del codice Java. Quello che viene fatto in questo metodo è il semplice controllo sulla lunghezza dell'ISBN. Presupponiamo infatti che questo codice debba essere lungo almeno 9 caratteri (è una validazione solo a scopo pedagogico). Nel caso in cui la stringa inserita dall'utente sia più corta di 9 caratteri verrà lanciata una ValidatorException, che conterrà il messaggio d'errore che vogliamo far ricevere all'utente. L'altro che abbiamo inserito serve per aggiungere il libro nel nostro database e per sapere se è possibile inserirlo oppure se è già presente un libro simile nella nostra biblioteca. Il metodo tuttavia restituisce sempre la stringa"aggiunto" pure se alla fine non effettua inserimenti su alcun database. La cosa ci interessa a noi è capire come poter utilizzare questa classe per l'inserimento dei dati e per la loro validazione, la memorizzazione dei dati sul database verrà affrontata in un altro articolo. Per definire il bean appena realizzato all'interno della nostra applicazione JSF, dobbiamo inserire alcuni parametri all'interno del file di configurazione faces-config.xml

```
<managed-bean>
    <managed-bean-name>Libro</managed-bean-
                                        name>
    <managed-bean-
    class>it.ioprogrammo.jsf.Libro</managed-bean-
                                        class>
    <managed-bean-scope>session</managed-
                                  bean-scope>
</managed-bean>
```

Questa entry definisce il nostro bean, specificando il nome con cui verrà utilizzato, la classe e lo scope. Quest'ultimo, come nelle applicazioni JSP/Servlet, definisce lo scope dell'oggetto, ovvero quanto un'istanza di questo oggetto deve rimanere in vita. Praticamente definendo lo scope come session, avremo istanziato un oggetto Libro per tutta la durata della nostra sessione. Per fare un esempio, un bean utile per tutta la sessione di un utente potrebbe essere una classe che simula un carrello elettronico, ovvero che mappa tutte cose che un utente intende acquistare.

FORM DI INSERIMENTO

Ora che abbiamo definito il bean che utilizzeremo per mappare le informazioni, possiamo vedere la pagina JSF che definisce il form d'inserimento

```
<%@ page contentType="text/html"%>
<%@ taglib uri="http://java.sun.com/jsf/core"
                                  prefix="f"%>
<%@ taglib uri="http://java.sun.com/jsf/html"
                                  prefix="h"%>
<f:view>

  <html>
    <head>
      <title>Biblioteca</title>
    </head>

    <body>
      <h:form>
        <h2>
          Inserimento libro
        </h2>
        <h4>
          Form di registrazione
        </h4>
        <table>
            <tr>
```

095-101 29-06-2007 16:52 Pagina 99


```
<td>Titolo:</td>
<td>
  <h:inputText value="#{Libro.titolo}"
                         required="true"
           id="titolo"/>
  <h:message for="titolo"/>
</td>
</tr>
<tr>
<td>Autori:</td>
<td>
  <h:inputText value="#{Libro.autori}"
                         required="true"
           id="autori"/>
  <h:message for="autori"/>
</td>
</tr>
<tr>
<td>Editore:</td>
<td>
  <h:inputText value="#{Libro.editore}"
                         required="true"
           id="editore"/>
  <h:message for="editore"/>
</td>
</tr>
<tr>
<td>ISBN:</td>
<td>
  <h:inputText value="#{Libro.isbn}"
                         required="true"
validator="#{Libro.validateISBN}" id="isbn"/>
  <h:message for="isbn"/>
</td>
</tr>
<tr>
<td>Prezzo:</td>
<td>
  <h:inputText value="#{Libro.prezzo}"
                         required="true"
           id="prezzo"/>
  <h:message for="prezzo"/>
</td>
</tr>
<tr>
<td>Data di pubblicazione:</td>
<td>
  <h:inputText
   value="#{Libro.pubblicazione}" required="true"
           id="pubblicazione">
     <f:convertDateTime pattern="dd-
                              MM-yyyy"/>
  </h:inputText>
  <h:message for="pubblicazione"/>
</td>
</tr>
</table>
<p><h:commandButton value="Insert"
                      action="insert" /></p>
   </h:form>
  </body>
 </html>
</f:view>
```

Analizziamo ora questa pagina. Utilizzando la taglib core, viene definita la vista principale, all'interno della quale caleremo i componenti che vogliamo visualizzare. Viene quindi realizzata una semplice table HTML, dove inseriamo i vari campi di input. Questi vengono realizzati utilizzando la jstl html di JSF. Come potete vedere per ogni campo di input viene inserito anche l'attributo del bean a cui è legato.

Inserimento libro

Form di registrazione

Titolo:
Autori:
Editore:
ISBN:
Prezzo: 0.0
Data di pubblicazione:

Insert

Fig. 3: Form d'inserimento

Inserimento libro

Form di registrazione

Titolo:
Autori:
Editore:
ISBN:
Prezzo: 0.0
Data di pubblicazione:

Insert

Fig. 4: Form d'inserimento con un errore

Inoltre in tutti i campi viene definito il parametro required come true. In questo modo decidiamo che vogliamo che vengano inseriti tutti i campi presenti nel form. Se l'utente non inserisce un campo e clicca sul pulsante per

l'invio, la richiesta arriva alla pagina JSF. Si seguono quindi tutte le fasi del ciclo di vita del framawork e nel momento in cui JSF si accorge che quel parametro richiesto non è stato inserito invia al browser la stessa pagina, inserendo vicino al campo vuoto la stringa di errore. Questo messaggio viene stampato grazie al tag <h:message for="PARAMETRO"/>. Questo tag serve per stampare un messaggio d'errore che ha come destinatario il parametro mancante o errato. Ci sono due campi di questo form che vengono trattati in maniera differente rispetto agli altri, data di pubblicazione e ISBN. Per quanto riguarda la data di pubblicazione, abbiamo imposto che la data sia nel formato dd-MM-yyyy, utilizzando il tag f:convertDateTime. L'ISBN invece viene validato attraverso il metodo validateISBN() che abbiamo definito precedentemente nel bean Libro. Per fare ciò basta specificare nell'attributo validator il nome del metodo che dobbiamo richiamare e questo verrà invocato nel momento in cui viene richiesta la validazione del form.
La action del bottone viene legata al metodo addLibro() del bean Libro che abbiamo definito. In questo modo sappiamo che quando il libro verrà aggiunto, riceveremo come risposta "aggiunto". Per definire l'ultimo passaggio, ovvero quello in cui dalla pagina di inserimento passiamo alla pagina di riepilogo, bisogna inserire un'altra entry nel file di configurazione

```
<navigation-rule>
    <from-view-id>/insert.jsp</from-view-id>
    <navigation-case>
      <from-outcome>aggiunto</from-outcome>
      <to-view-id>/aggiunto.jsp</to-view-id>
    </navigation-case>

    <navigation-case>
      <from-outcome>nonaggiunto</from-
                                       outcome>
      <to-view-id>/nonaggiunto.jsp</to-view-id>
    </navigation-case>
</navigation-rule>
```

Grazie a questa regola di navigazione sappiamo quando il nostro libro è stato aggiunto alla nostra biblioteca, richiamando il metodo addLibro() della classe Libro. In questa applicazione abbiamo definito che la risposta sia sempre "aggiunto", ma in un caso reale dovremo inserire il bean nel database e verificare se tutto è andato bene. Per avere un'altra conferma dell'esecuzione del metodo possiamo leggere il log di Tomcat, dove troveremo la scritta "Libro aggiunto alla libreria". Per il riepilogo dei dati inseriti basta la pagina che riportiamo di seguito

```
<%@ page contentType="text/html"%>
<%@ taglib uri="http://java.sun.com/jsf/core"
                              prefix="f"%>

<%@ taglib uri="http://java.sun.com/jsf/html"
                              prefix="h"%>

<f:view>
  <html>
    <head>
      <title>Biblioteca</title>
    </head>
    <body>
      <h:form>
        <h4>
          Libro inserito
        </h4>
        <table>
          <tr>
            <td>Titolo:</td>
            <td>
              <h:outputText value="#{Libro.titolo}"
                                              />
            </td>
          </tr>
          <tr>
            <td>Autori:</td>
            <td>
              <h:outputText value="#{Libro.autori}"
                                              />
            </td>
          </tr>
          <tr>
            <td>Editore:</td>
            <td>
              <h:outputText
                     value="#{Libro.editore}" />
            </td>
          </tr>
          <tr>
            <td>ISBN:</td>
            <td>
              <h:outputText value="#{Libro.isbn}"
                                              />
            </td>
          </tr>
          <tr>
            <td>Prezzo:</td>
            <td>
              <h:outputText
                     value="#{Libro.prezzo}" />
            </td>
          </tr>
          <tr>
            <td>Data di pubblicazione:</td>
```

```
            <td>
              <h:outputText
                     value="#{Libro.pubblicazione}" >
                   <f:convertDateTime pattern="dd-
                                     MM-yyyy"/>
              </h:outputText>
            </td>
          </tr>
        </table>
      </h:form>
    </body>
  </html>
</f:view>
```

Essendo il bean definito con scope session, è ancora presente nella nostra sessione, quindi lo utilizziamo per effettuare il riepilogo con una semplice tabella.

LISTA DEI LIBRI

Siamo arrivati all'ultimo pezzo della nostra applicazione, quello in cui stampiamo a schermo tutte le informazioni relative ai libri della nostra biblioteca. Per gestire informazioni all'interno di JSF abbiamo a disposizione UIData, un componente per gestire dati in forma tabulare.

```
<h:dataTable value="#{biblioteca.getLibri}"
                                      var="lista">
  <h:column>
    <f:facet name="header">
      <h:outputText value="Titolo" />
    </f:facet>
    <h:outputText value="#{lista.titolo}" />
  </h:column>

  <h:column>
    <f:facet name="header">
      <h:outputText value="Autori" />
    </f:facet>
    <h:outputText value="#{lista.autori}" />
  </h:column>

  <h:column>
    <f:facet name="header">
      <h:outputText value="Editore" />
    </f:facet>
    <h:outputText value="#{lista.editore}" />
  </h:column>

<h:column>
    <f:facet name="header">
      <h:outputText value="ISBN" />
    </f:facet>
    <h:outputText value="#{lista.isbn}" />
  </h:column>

<h:column>
    <f:facet name="header">
      <h:outputText value="Prezzo" />
    </f:facet>
    <h:outputText value="#{lista.prezzo}" />
  </h:column>

</h:dataTable>
```

All'inizio, nella definizione di questo tag, abbiamo usato una classe, biblioteca, che dovrebbe restituirci una List (java.util.List) di oggetti Libro. Qui non abbiamo definito questa classe, anche perché non è stato definito l'accesso verso il database, comunque non è difficile immagine una classe che si connette al nostro database e recupera tutti i libri che sono presenti, incapsulandoli in oggetti Libro. Il tag quindi, una volta ottenuta questa lista, non fa altro che elaborare tutto il suo contenuto, stampando una tabella con tutti i libri. Quello che viene fatto è definire semplicemente il nome della colonna e successivamente associarle un valore del classico oggetto Libro che vogliamo visualizzare.

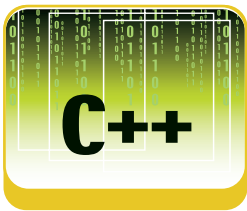
CONCLUSIONI

In questo articolo abbiamo visto come poter utilizzare il ciclo di vita del framework JSF per poter modellare tutto quello che accade nella nostra applicazione. Una cosa utile da approfondire è il tema relativo all'immediate event, un meccanismo che permette di definire un evento immediato (AJAX style) per poter gestire degli eventi senza il bisogno di effettuare una richiesta. Le librerie che JSF mette a disposizione per gestire la visualizzazione dei dati sono diverse, inoltre è possibile realizzare dei componenti aggiuntivi a quelli standard.
L'utilizzo di JSF garantisce comunque una separazione netta fra logica di business, dati e view dell'applicazione.
Questo approccio favorisce, ovviamente, la manutebilità del software.
La difficoltà è tutta inizale, quando è necessario superare il gradino di conoscenza che porta a comprendere il ciclo di vita di un'applicazione JSF. Una volta compiuto questo sforzo i successivi passi risultano decisamente agevoli ed un corretto utilizzo del pattern MVC mostra tutti i suoi benefici durante lo sviluppo di applicazioni dalle dimensioni generose.

Federico Paparoni

L'autore, Federico Paparoni, può essere contattato per suggerimenti o delucidazioni all'indirizzo email federico.paparoni@javastaff.com

C++ SCOPRE LE FINESTRE!

COME SI CREA UNA FINESTRA IN C++? UNA SEMPLICE DOMANDA APRE UN UNIVERSO DI RISPOSTE E DI LIBRERIE. SCOPRIAMO QUALI SONO E INCOMINCIAMO A SVILUPPARE INTERFACCE GRAFICHE SUPERPERFORMANTI E MULTIPIATTAFORMA

Benvenuti ad una nuova serie di questa rubrica, in cui ogni mese tentiamo di "migliorare" il C++, coi più efficaci strumenti a nostra disposizione. Nonostante sia un linguaggio eccellente, infatti, il C++ non è perfetto, soprattutto in un punto: *la libreria standard è ancora troppo carente*.

Nelle scorse puntate abbiamo visto come espanderla per la gestione automatica della memoria (grazie agli smart pointer e ai garbage collector) e per l'interpretazione avanzata del testo (grazie agli strumenti forniti da boost), ma rimangono ancora diversi punti scoperti.

In particolare, una delle domande più frequenti che ricevo per e-mail è questa: "Come si crea una finestra in C++?". La cosa è piuttosto disarmante, perché dà per scontato che i linguaggi "moderni" dispongano di un sistema (o più!) per rappresentare finestre, pulsanti e componenti – tutto ciò che un po' più professionalmente va sotto il nome di *Graphical User Interface* (GUI).

Bisogna innanzitutto chiarire che quest'idea *è sbagliata*: molti linguaggi moderni e professionali non dispongono nativamente di una GUI, e per buone motivazioni che vedremo. Ma d'altro canto è innegabile che, oggi, la programmazione grafica sia diventata una componente essenziale di una larga fetta di applicazioni mainstream.

Premesso questo, resta ancora in sospeso la domanda "Come si crea una finestra in C++?". Risponderò nel modo più sintetico: "*Non si può*. Non col solo C++ standard, quantomeno". Ma - come al solito - quando c'è una limitazione di questa portata, ecco che i programmatori si sbizzarriscono a scrivere librerie per superare l'ostacolo.

In quest'articolo vedremo perché il C++ non ha una libreria GUI standard, quali sono quelle disponibili, e incoroneremo "La Migliore", dedicando il resto di questa serie ad una sua introduzione. Per stilare una "classifica" in un campo così complesso dobbiamo chiederci: "Come dovrebbe essere una libreria GUI, per poter essere ammessa nel C++ standard?".

CROSS PLATFORM

Uno dei problemi di una libreria GUI standard è innanzitutto politico (credevate che la politica non c'entrasse?). Al mondo non esiste uno standard ufficiale per la gestione delle finestre e dei componenti grafici, ma ogni Sistema Operativo incorpora un suo window system (come il caso del Mac OS o di Windows Vista), oppure si basa su sistemi esterni (come X Window System). Esistono quindi decine di set di API, componenti e modelli di programmazione diversi e incompatibili fra loro.

Una libreria standard non può assolutamente sceglierne uno (o un paio) e tagliare fuori tutti gli altri: viceversa deve poter girare indipendentemente dalla piattaforma utilizzata, in maniera assolutamente cross platform, e questo è un vincolo imprescindibile. Un programmatore deve poter svolgere il suo lavoro tranquillamente, sicuro di poter poi compilare il suo programma su ogni sistema.

Realizzare una libreria del genere pone dei grossi problemi, e infatti la maggior parte dei tentativi si è indirizzata sui singoli Sistemi Operativi (spesso sui singoli modelli di un Sistema Operativo). Questo, purtroppo, taglia fuori anche soluzioni per altri versi molto promettenti come *SmartWin++*, libreria avanzatissima per concezione e sviluppo, ma estremamente limitata dalle sue finalità windows-only.

Allo stesso modo, la totalità delle librerie sfornate da Microsoft in questi anni (*MFC, Windows Forms, ATL, WTL*, ecc…) non consente di valicare i confini dei sistemi operativi Windows, e/o del Framework .NET.

Scrivere un sistema GUI cross-platform non è, però, un'impresa impossibile: librerie come

WxWidgets, Qt, GTK+ e FLTK permettono (chi più chi meno) di scrivere codice che potrà essere compilato per molte piattaforme differenti, con poche (o nessuna) modifica.

NATIVA

Abbiamo già eliminato decine di contendenti al trono, stabilendo che La Libreria dovrà essere cross-platform. Ora dobbiamo chiederci come possa una libreria sola superare l'ostacolo di decine di API e modelli differenti. Ci sono due sistemi fondamentali, in competizione fra loro.
Il metodo dell'*emulazione* consiste nel fondare la libreria su un solo modello, su un unico insieme di componenti, scegliendolo fra quelli realmente esistenti (ad esempio, Win32), oppure inventandolo di sana pianta (il che è molto più democratico. Ricordate i problemi politici?). L'aspetto dei componenti e il loro funzionamento viene quindi emulato sulle varie piattaforme di destinazione, spesso con la possibilità di usare temi differenti che si avvicinino all'aspetto nativo delle applicazioni su quella piattaforma/modello.
Questo sistema permette di usare un'unica architettura software coerente, e rende molto più semplice il porting fra i vari sistemi e modelli.
Il metodo opposto è quello nativo, nel quale, invece, vengono usati proprio i componenti originali di ciascun sistema operativo, senza alcuna emulazione. Questo sistema richiede un grande lavoro per riuscire ad ottenere un interfaccia astratta di base, capace di descrivere a priori il funzionamento di componenti che sono in sé spesso molto diversi – un lavoro che talvolta è impossibile. In una libreria nativa, ad esempio, può capitare di avere a disposizione un certo oggetto o una certa funzione membro soltanto sotto Windows, ma non sotto Mac OS (e viceversa, ovviamente).
Ma anche il sistema nativo offre dei grandi vantaggi. Ad esempio, non essendoci alcuno "strato" di emulazione intermedio, i componenti rispondono in maniera più rapida e precisa – spesso è impossibile "emulare" efficacemente alcune caratteristiche gelosamente custodite dal Sistema Operativo.
Ma il vantaggio principale di usare un toolkit nativo è soprattutto quello di offrire all'utente finale un'applicazione che si comporti in maniera per lui "*naturale e coerente*". Questo è un punto fondamentale, che possono apprezzare anche coloro che conoscono un solo sistema operativo. Windows XP, ad esempio, ha una schermata tipica per la selezione di uno o più file, come quella mostrata in figura 1. Gli utenti XP ci mettono un po' ad impratichirsi, ma poi quella schermata diventa loro familiare. In figura 2, un pessimo programmatore ha avuto la bella idea di definire una sua schermata "emulata" per la stessa, identica, funzione. Questa spiazza l'utente, che deve spendere inutilmente tempo ed energie per capirne il funzionamento.Per questa serie di vantaggi e svantaggi, il primato fra interfacce native ed emulate si è trasformato in una delle tante Guerre Sante che popolano l'informatica. Alla fazione emulatrice appartengono le librerie *GTK+ e Qt* (che quantomeno prevede dei temi per "mascherare" l'emulazione), a quella nativa toolkit come *wxWidgets*. (Per la cronaca, alla luce del rapporto costi/benefici di cui sopra, io sono schierato al fianco dei nativi.)

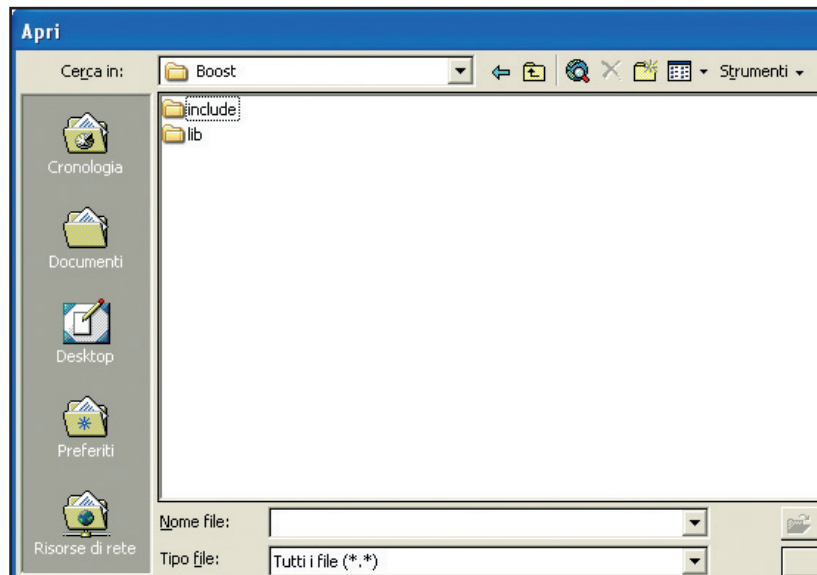
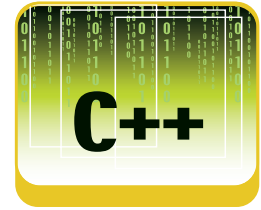



Figura 1: Schermata di caricamento file "nativa" in Windows XP

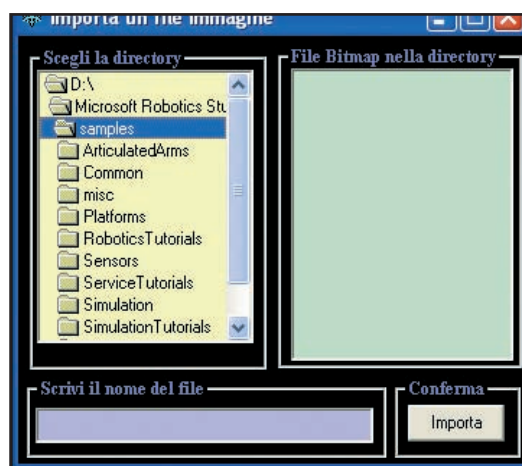
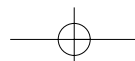
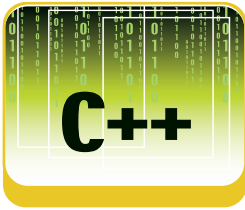



Figura 2: (Orrenda) schermata di caricamento file "emulata" in Windows XP

LEGGERA

Uno dei principi cardine del C++ è che: "*non bisogna pagare ciò che non si usa*". Una libreria, in effetti, può essere "pagata" in diverse valute: riduzione delle prestazioni, complicazione inutile del codice, ingrassamento dei file eseguibili.
Una GUI standard per il C++ non potrebbe permettersi nessuno di questi vizi: molto spesso, infatti, i programmatori non vorranno utilizzare affatto le funzionalità grafiche – in effetti, molti di loro sviluppano per un sistemi a riga di comando, e altri per macchine che non hanno nemmeno uno schermo!
Una buona libreria non stravolgerà mai la struttura del programma, cambiando ad esempio il punto d'ingresso dell'applicazione, o le routine di allocazione della memoria.
In quanto a leggerezza la palma d'oro va a *FLTK*, una libreria statica, scritta col preciso scopo di ottenere eseguibili leggeri, veloci e privi di dipendenze. Altrettanto leggera è *SmartWin++*, altra libreria (statica) scritta con lo stesso scopo, ma molto meglio progettata.
Il fatto stesso che l'esigenza di leggerezza abbia portato alla scrittura di ben due librerie apposite, fa capire che le "grandi" (wxWidgets, Qt, GTK+) se la cavano molto male sotto questo punto di vista. Oltre ai problemi già gitati, queste librerie spesso incorporano molte funzionalità che non c'entrano nulla con le interfacce grafiche - dal parsing XML al supporto per ODBC, dal threading ai socket – che possono risultare utili ma che, in mia opinione, dovrebbero essere rilasciati come librerie esterne. Questa convergenza contribuisce invece ad appesantire inutilmente le librerie fino a farle diventare dei *framework* mastodontici.

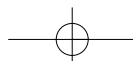
LIBERA

Uno dei grandi pregi del C++ è che si tratta di uno dei pochi linguaggi non-proprietari utilizzati nel mercato mainstream. Pertanto una componente della libreria standard dovrebbe essere altrettanto libera da vincoli economici e proprietari.
Sotto questo punto di vista, al contrario degli altri, siamo messi mediamente molto bene – almeno trascurando il mondo Microsoft, che abbiamo già accantonato per abbracciare i modelli cross-platform. Tutte le soluzioni degne di nota sono gratuite e libere, rilasciate sotto licenza LGPL, o modelli con vincoli ancora più laschi.
L'unica eccezione è la libreria Qt - una delle più valide e significative. Qt è open source, ma è e resta di proprietà della TrollTech, che la rilascia sotto un "dual licensing". In altre parole, è possibile utilizzare gratuitamente la libreria Qt soltanto se si sviluppa un programma open-source. Per realizzare un normale programma commerciale a sorgente chiuso, invece, è richiesto un pagamento che va dai 1420 della versione base (console) ai 2630 di quella più avanzata (desktop).

MODERNA

Una libreria standard C++ deve essere scritta in C++! Sembrerebbe tanto ovvio da essere inutile specificarlo, invece il panorama attuale non è molto confortante. I problemi sono due:

- **Scrivere una libreria GUI è un compito complesso** sotto il profilo del design OOP. Ad esempio, sono richiesti particolari pattern per la gestione dei callback (i cosiddetti "eventi") che devono conciliare rapidità di esecuzione, concisione del codice e sicurezza a tempo di compilazione.
- **Le librerie più importanti e collaudate sono nate nei primi anni novanta**, prima che il C++ incorporasse gran parte della libreria standard attuale, e prima che venissero studiate e scoperte le tecniche di programmazione relative alle parti più "avanzate" del C++.

Il risultato è che spesso queste librerie non sono realmente scritte in C++. FLTK, ad esempio, appartiene a quelle librerie nate sotto la concezione (sbagliata) che "leggero" e "portabile" voglia dire "privo di RTTI, template, eccezioni, namespaces, e un buon 80% delle feature del C++". Quest'idea (riassunta nella Mozilla C++ Compatibility Guide) è oggi considerata obsoleta e ormai ridicola, e rende FLTK inutilizzabile da ogni programmatore C++ amante del linguaggio. GTK+, invece, è scritto direttamente in C – ma esiste una libreria wrapper chiamata **GTKmm** (che sta per GTK--), che gode di una buona implementazione. Il C++ usato da WxWidgets e Qt, infine, lascia molto a desiderare. Entrambi, infatti, sono stati costruiti sulla stessa concezione di FLTK, e fanno largo uso di **macro** e **puntatori void***, l'antitesi della buona programmazione in C++, che dovrebbe essere sempre robusta e sicura rispetto ai tipi. Mentre WxWidgets si limita a scomodare il preprocessore standard, Qt implementa un suo sistema per l'RTTI per

il quale usa addirittura un preprocessore dedicato chiamato moc, rendendo così i programmi scritti per questa libreria non-standard. Oltre ad essere estraneo alle fondamentali norme di stile (e di standard), l'uso estensivo di macro e del preprocessore spesso impedisce ai parser di svolgere efficacemente il proprio lavoro: è noto che gli analizzatori del codice hanno seri problemi a trattare programmi scritti con wxWidgets e Qt, e anche IDE sofisticati come Visual Studio tendono a confondersi, inibendo funzionalità come l'IntelliSense. Inoltre, molte librerie di questo tipo (proprio perché nate prima della standardizzazione della STL) si affannano ad includere librerie per i contenitori alternative a quelle standard. Ecco nascere oggetti String, Vector, supporti RTTI, eccetera, che nei migliori dei casi ricalcano le implementazioni dei componenti già esistenti nel C++ (e nei peggiori, espongono delle funzioni membro orripilanti, alla *String::Printf*). La libreria che si aggiudica il premio per il miglior design è invece **SmartWin**++, sviluppata secondo le tecniche consolidate negli ultimi anni, ovverosia in quello che viene chiamato comunemente – dal titolo del testo di Alexandrescu "*Modern C++ Design*" – **C++ moderno**. Dal punto di vista dello stile, SmartWin può essere considerata un'ottima base per la creazione di una GUI C++ standard.

UNA GUI MATURA

Una GUI standard per il C++ non dovrebbe mai soffrire problemi di scarsa affidabilità: pulsanti che saltano inspiegabilmente, giochi di trasparenze che non traspaiono, sottomenù che vanno in crash... Questi, ovviamente, sono dettagli implementativi la cui responsabilità sarebbe lasciata interamente sulle spalle degli scrittori di compilatori, e non farebbero parte dell'interfaccia standard.
Tuttavia l'affidabilità resta una componente essenziale per la scelta di una delle librerie realmente esistenti! Come spesso accade nel mondo dell'open-source, le librerie più stabili sono quelle storiche, che sono state sulla scena per più tempo e quindi hanno avuto più possibilità di bug-report ed evoluzioni, nonché di stabilire una solida user-base e una comunità di supporto. Da questo punto di vista, WxWidgets e Qt, sono e restano le librerie più mature ed affidabili sulla piazza. Vantano inoltre una nutrita community di utilizzatori per cui recuperare la documentazione in rete è piuttosto semplice.

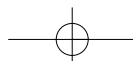
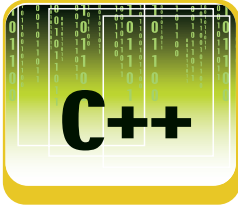
Libreria	Cross Platform	Nativa	Leggera	Libera	Stile C++	Matura
WxWidgets	Molto	Sì	No	Sì	Vecchio	Molto
Qt	Sì	No	No	No	Vecchio e non-standard	Molto
GTK+	Sì	No	No	Sì	C (C++ con GTKmm)	Sì
SmartWin++	No	Sì	Sì	Sì	Moderno	No
FLTK	Sì	No	Sì	Sì	Vecchio	Poco

Tabella 1: Confronto tra diverse librerie GUI, secondo i sei criteri discussi.

TIRIAMO LE SOMME

A questo punto dovrebbe essere chiaro il motivo per cui ho speso tanto tempo ed energie in un'analisi sommaria delle caratteristiche di una Libreria Ideale: per illustrare come, al di là delle Guerre Sante e delle preferenze personali, **non esista alcuna libreria palesemente migliore delle altre**. Come mostra la **Tabella 1**, tutte hanno pregi e difetti, e soprattutto nessuna è contemporaneamente **cross-platform, nativa, libera, leggera, moderna** e **affidabile**. Il che fa capire quanto scrivere una buona libreria grafica adatta ad ogni tipo di situazione (dal programma minimo a quello con decine di form e centinaia di controlli) sia un'impresa ai limiti dell'impossibile, e spiega perché il C++ standard non prevede (e probabilmente non prevederà mai) una libreria GUI.
Dovendo eleggerne una per illustrarla in questa serie, qui opterò per WxWidgets, essendo una di quelle che più si avvicina al modello ideale che ho tracciato in queste pagine. La concorrente più diretta (Qt), infatti ha qualche porting in meno, non è nativa, forza l'uso di un C++ non-standard e soprattutto è proprietaria, con costi non trascurabili per realtà commerciali medio-piccole.
Ciò non significa, tuttavia, che wxWidgets sia perfetta; come abbiamo già accennato, ha i suoi difetti di stile e di implementazione, che approfondiremo nel corso della serie.
Se ci è concesso sognare, l'ideale sarebbe una libreria con i pregi di WxWidgets e la leggerezza/modernità di SmartWin++ - e questo è un suggerimento per chiunque si senta tanto temerario da voler scrivere The Next Big Thing nel mondo C++.

I PORT DI WXWIDGETS

Per capire wxWidgets occorre comprenderne la struttura, che – come gran parte delle librerie cross-platform – è fatta a strati sovrapposti.
L'API wxWidgets pubblica rappresenta lo

"strato" esterno, ed è costituita da un insieme di classi e funzioni che forma l'interfaccia di alto livello. Il programmatore finale scrive il codice secondo quest'interfaccia, astraendosi così dall'implementazione sottostante e dai diversi Sistemi Operativi. **I port wxWidgets** sono le implementazioni dell'API pubblica costruite basandosi su una specifica **Platform API** (come win32, GTK+, Cocoa...), che a sua volta comunica direttamente con le API esposte dal Sistema Operativo. Uno dei vantaggi di usare wxWidgets e che i port sono veramente tanti. Ecco i principali:

Per Windows

- **wxMSW**, costruito direttamente sulle API Windows, permette di scrivere codice per tutte le piattaforme Windows a 32 e 64 bit: da Windows 95 fino a Vista.

- **wxWinCE**, è una costola un po' ridotta di wxMSW, scritta per Windows CE (e che quindi permette di portare i propri programmi su aggeggi come pocket PC e smartphone).

Per Linux

- **wxGTK**, costruito su GTK+, è la scelta più indicata per la programmazione su linux/unix. Sottolineo il fatto che wxWidgets, quindi, "include" GTK+, e non è direttamente in competizione con esso. Grazie a wxWidgets, infatti, è possibile usare la stabilità di GTK+ sotto linux, senza rendere il sistema instabile ed emulato in caso di porting verso altri SO.

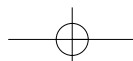
- **wxX11 / wxMotif**, costruiti su Xlib e Motif (e varianti), sono delle possibilità per coloro a cui non piace GTK+ o per sistemi su cui non è possibile montarlo. L'attrattiva di queste API è molto scarsa, e i porting non molto sviluppati, pertanto consiglio caldamente di considerare GTK+, se possibile. Un quantitativo industriale di applicazioni Linux utilizza oggi questa tecnologia

Per Macintosh

- wxMac, costruito sulle API Carbon, permette di scrivere per Max OS 9, Mac OS X, e anche per Mac OS 8.X grazie alle API di Classic.

- wxCocoa, costruito sulle API Cocoa di Mac OS X. È ancora in fase di sviluppo, pertanto consiglio caldamente di compilare su wxMac con CodeWarrior o con gli Apple tools.

CONFIGURAZIONI DI WXWIDGETS

Prima di parlare dell'installazione di wxWidgets (ci arriviamo, lo giuro!), dobbiamo per forza dire due parole sulle sue configurazioni. WxWidgets, infatti, può essere compilata con molte opzioni diverse, tutte combinabili fra loro fino ad ottenere infinite permutazioni. Qui vediamo le quattro scelte più importanti, che di solito definiscono una specifica "configurazione":

Release o Debug: Questa è semplice, e immagino ve l'aspettavaste: come al solito, nella versione debug sono attive asserzioni, controlli, eccetera. Se siete dei novizi (ma anche no), è *obbligatorio* usare la versione **debug**. Quella release andrebbe creata solo il giorno prima della consegna al cliente.

Statica o Dinamica: Anche questa è un classico. WxWidgets può essere inglobata direttamente nel vostro eseguibile (file .lib o .a), oppure essere condivisa esternamente (file .dll o .so). Per evitare il dll Hell, consiglio caldamente la versione **statica**.

Unicode o Non-Unicode: Questa forse non ve l'aspettavate. WxWidgets può essere compilato con il supporto per i caratteri estesi o meno. Dal punto di vista della programmazione non cambia nulla, perché userete delle macro (come _T) che spostano il discorso a un livello più alto (per quanto è possibile usando una macro). Il mio consiglio è di usare la versione **Unicode**.

Multilibreria o Monolitica: Come ho già accennato nel paragrafo "leggerezza", wxWidgets è un vero e proprio framework più che una libreria, e incorpora tante funzionalità che non hanno niente a che fare con la programmazione grafica – funzionalità che probabilmente non userete mai. È ingiusto pagare per ciò che non si usa, pertanto potete scegliere di spezzare la libreria in più sottolibrerie, ciascuna con una sola responsabilità. Vi consiglio di seguire il consiglio, abbandonando il monolite a favore della versione **multilibreria**.

Questo discorso tornerà utile nel prossimo paragrafo. Ricordatevi che i file che creerete avranno questa forma: wx[libreria][versione][u?][d?]. Se usate la versione 2.8 (quella attuale), e producete la libreria base con supporto debug e unicode, avrete un file wxbase28ud.

COMPILARE WXWIDGETS

"Va bene, d'accordo, ci hai convinto, wxWidgets è bello, gira su tante piattaforme e ha tante configurazioni. Ma come si fa funzionare 'sta roba?". Se vi state chiedendo questo, è giunto il momento di parlare dell'installazione, della compilazione e soprattutto della configurazione di wxWidgets – momento che sarà uno dei più delicati perché queste operazioni non sono ancora il massimo della semplicità, purtroppo. Partiamo dalla compilazione. Dividerò il discorso in due vie fra le molte possibili: una per Windows (con Visual C++ Express 2005) e una per Linux/Unix/MacOSX (con GCC).

INSTALLAZIONE SU WINDOWS

Scaricate il **Visual C++ Express 2005**, dall'indirizzo http://msdn.microsoft.com/vstudio/express/downloads/, e installatelo.
Scaricate il **Windows Server 2003 Platform SDK R2 Full**, dall'indirizzo http://www.microsoft.com/downloads, e installatelo.
Configurateli seguendo i semplici passi indicati all'indirizzo http://msdn.microsoft.com/vstudio/express/visualc/usingpsdk/.
Scaricate **wxMSW** dall'indirizzo www.wxwidgets.org/downloads, e installatelo.
Aprite il file ...\build\msw\wx.dsw. Vi verrà chiesto di convertire i progetti nella nuova versione di Visual Studio. Rispondete "Yes to All".
Andate nel menù *Build->Configuration Manager*, e scegliete dal menù a tendina la configurazione che volete. Se volete seguire i consigli del paragrafo precedente, scegliete "**Unicode Debug**". Avviate la compilazione premendo F7, e aspettate (la compilazione e lunga e il compilatore è buono ma lento), ignorate eventuali warning di tipo "*deprecated*". Nella cartella *\lib\vc_lib* troverete tutte i file *.lib* compilati, nominati secondo lo schema visto nel paragrafo precedente.

INSTALLAZIONE SU LINUX

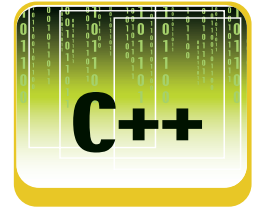
Scaricate **wxGTK** dall'indirizzo www.wxwidgets.org/downloads e decomprimetelo in una directory.
Create al suo interno una directory (ad esempio: build28ud), e spostatevi al suo interno.
Lanciate *configure*, passandogli i parametri in accordo con le vostre opzioni di configurazione. Ad esempio, per quelle consigliate nel paragrafo precedente:

```
../configure --enable-unicode --enable-debug --
                    disable-shared --disable-monolithic
```

Lanciate *make* (sempre stando in build28ud), per costruire le librerie:

```
make
```

Lanciate *make install* per copiare i file in usr/local (alterate il prefisso se volete installarli in un'altra locazione):

```
make install
```

CONFIGURARE L'IDE

Se avete raggiunto questo paragrafo, complimenti! Avete ora a disposizione una bellissima installazione di wxWidgets (probabilmente in modalita debug, unicode, multilibreria, statica).
Ora non ci resta che chiudere questo primo appuntamento introduttivo creando la nostra prima applicazione col nostro IDE di fiducia.
Anche qui, dividerò il discorso in due, sotto windows col solito **Visual Studio 2005 Express** (che ormai do per scontato abbiate installato) e sotto linux con **Kdevelop**, probabilmente l'IDE più completo fra quelli del Pinguino.
Nota: chiamerò "WXDIR" la directory in cui avete installato wxWidgets.

su Windows
Create un nuovo progetto di tipo Empty Project.

Aggiungete un nuovo file al progetto
Andate sul menu Project -> Properties.
Scegliete *C/C++ -> General*, e nel campo "*Additional Include Directories*" inserite: **WXDIR\include; WXDIR\lib\vc_lib\mswud**.

Scegliete *C/C++ -> Preprocessor*, e nel campo "*Preprocessor Definitions*:" inserite: **WIN32, __WXMSW__, _WINDOWS, _DEBUG, __WXDEBUG__**

Scegliete C/C++ -> *Code Generation*, e nel campo "*Runtime library*" selezionate: **Multithreaded Debug DLL**.

Scegliete *Linker -> General*, e nel campo

FEDERICO PAPARONI
può essere contattato per suggerimenti o delucidazioni all'indirizzo email federico.paparoni@javastaff.com

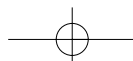
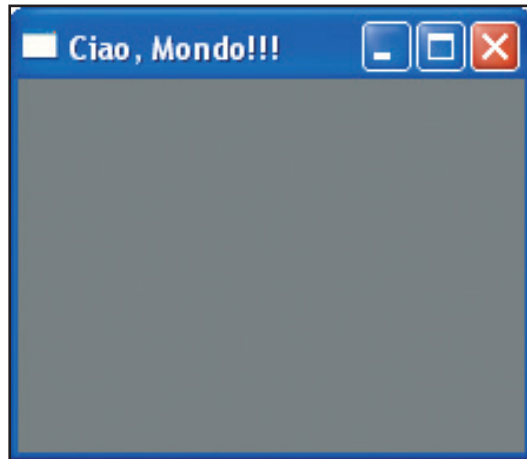
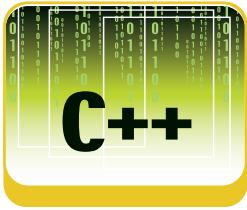



Figura 3: L'applicazione di test come appare sotto Windows.

"*Additional Library Directiories*" inserite: **WXDIR\lib\vc_lib**

Scegliete *Linker -> Input*, e nel campo "*Additional Dependencies*" inserite le librerie che vi servono (le vedremo nei prossimi appuntamenti). Per ora inserite quelle di base: **wxbase28ud** e **wxmsw28ud_core.lib**.
Scegliete *Linker -> System*, e nel campo "Subsystem" selezionate: Windows.

Aggiungete alle *additional dependencies* anche le librerie windows che vi serviranno. Per un programma minimo: **kernel32.lib, user32.lib, gdi32.lib**

In caso abbiate comunque problemi di linking, copiate le impostazioni degli esempi nella cartella samples.

su Linux
Andate in usr/local/ (o dove avete installato wxWidgets) e lanciate wx-config per ottenere i flag di compilazione e di linking:

```
wx-config --cxxflags
wx-config --libs
```

e prendetene nota (vi serviranno più avanti).
Installate KDevelop manualmente, oppure (molto meglio) col vostro gestore di pacchetti preferito (meglio), e avviatelo.
Create un Nuovo Progetto.
Sotto "C++" scegliete *Simple Hello World Program* (lasciate perdere il wizard wxWidgets).
Cancellate ogni file già creato da KDevelop.
Aprite le Project Options e scegliete Configure Options.
Cliccate sulla linguetta C++ e incollate i risultati ottenuti con wx-config --cxxflags.
Cliccate sulla linguetta *General* e incollate i risultati ottenuti con wx-config --libs
Aggiungete un nuovo file all'automake manager (main.cpp, vedi prossimo paragrafo).
Cliccate su *Build -> Run Automake & Friends*
Cliccate su *Build -> Run Configure*

L'APPLICAZIONE DI TEST

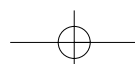
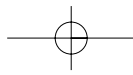
Ecco il file *main.cpp*, da inserire nei vostri progetti (vedi paragrafo precedente):

```
//L'Hello World in WxWidgets!
#include "wx/wx.h"

class MyApp : public wxApp
{
public:
    virtual bool OnInit();
};

IMPLEMENT_APP(MyApp)

bool MyApp::OnInit()
{
    if ( !wxApp::OnInit() )
        return false;

    wxFrame* frame = new wxFrame(0,
                 wxID_ANY, _T("Ciao, Mondo!!!"));
    frame->Show(true);

    return true;
}
```

Compilatelo e fate partire il debug con il vostro IDE di fiducia. Se avete svolto tutti i passaggi alla lettera, il programma funzionerà e otterrete una finestra del simile a quella in Figura 3 (sotto Windows). Complimenti! Siete ufficialmente pronti ad entrare nel magico mondo di wxWidgets.

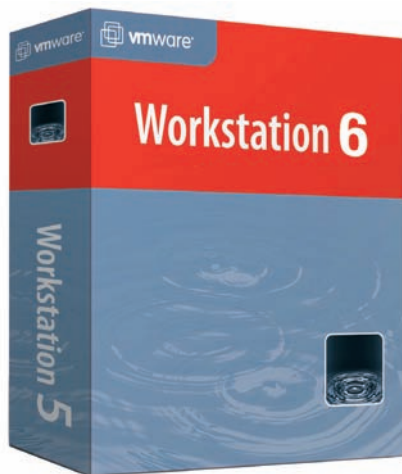
CONCLUSIONI

Come "introduzione" abbiamo davvero fatto molto, partendo dalle caratteristiche delle librerie grafiche e arrivando all'installazione e alla configurazione degli IDE. È stato un po' laborioso, ma necessario. A partire dal mese prossimo, infatti, potremo iniziare a fare "sul serio" studiando il framework e sporcandoci le mani con righe e righe di codice.

Roberto Allegra

SOFTWARE SUL CD

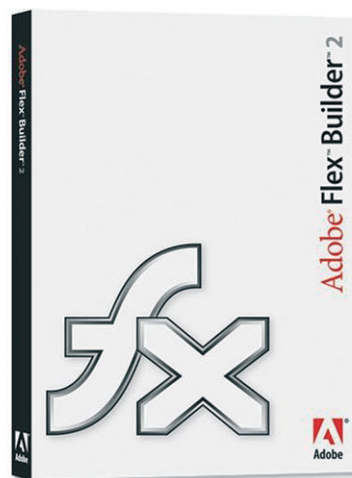
VMWARE WORKSTATION 6.0

L'ARTE DEL VIRTUALE

Non c'è dubbio che la virtualizzazione sia una delle nuove frontiere verso cui si sta direzionando lo sviluppo del software. Virtualizzare significa disporre di un sistema operativo completo in un qualche modo innestato nel sistema operativo principale eppure facente parte di una sandbox del tutto disconnessa da esso.

Inutile dire che i vantaggi di questa tecnica sono innumerevoli. Prima di tutto gli sviluppatori possono provare il comportamento dei propri programmi su più sistemi operativi senza per questo dover disporre di tante macchine quanti sono gli os attualmente disponibili oppure dover formattare continuamente. Ci sono poi tante e tante situazioni in cui una macchina virtuale risulta utilissima ad esempio quando si vuole isolare un programma potenzialmente dannoso per studiarlo, oppure quando si vogliono provare più distribuzioni di Linux, o semplicemente quando si vogliono fare delle prove senza sporcare il proprio sistema. Uno dei leader nel campo della virtualizzazione è proprio Vmware di cui vi presentiamo la versione Workstation 6.0 del proprio virtualizzatore. Le news introdotte sono interessanti, si va dal supporto verso Vista fino ai 64 bit. Per potere utilizzare pienamente questo prodotto per 30 giorni è necessario registrarsi presso il sito www.vmware.com/go/winit6 ed ottenere un codice di sblocco. Il codice in questione è valido per 30 giorni alla scadenza dei quali bisognerà procedere ad acquistare una versione completa

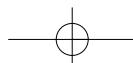
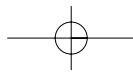
ADOBE FLEX BUILDER 2.0

FLASH CON QUALCOSA IN PIÙ

Si sa, i designer per anni hanno amato flash e tutt'ora lo amano. Poter costruire siti interattivi senza doversi sentire limitati dall'html e senza doversi preoccupare della piattaforma è decisamente un'opportunità golosa per qualunque web designer. Se a tutto ciò si aggiunge che Flash consente la realizzazione di animazioni veramente complesse con pochissimo sforzo si capisce perché davanti a certe opere si rimanga a bocca aperta.

Dove sono dunque i limiti di flash? da un punto di vista di un programmatore il problema è una scarsa interazione con i database, un linguaggio che comunque diverge dalle tecniche classica a cui siamo abituati. Ed ecco che arriva Flex. Meno grafica, meno ambienti dedicati ai designer, più programmazione, più interazione con i database, più strutture complesse, più XML. E' proprio XML la chiave di volta di Flex. Si crea un file XML che rappresenta la struttura di un documento lo si da in pasto al compilatore e quello tira fuori un file swf. A fare da contorno a tutto ciò c'è eclipse. Infatti per agevolare l'uso dell'SDK, Adobe ha sviluppato un plugin per il noto eclipse. L'accoppiata è di quelle vincenti. Il solo SDK di Flex è gratuito, il prodotto completo integrato in eclipse ha invece un valore commerciale di 449 € più IVA

ALFRESCO COMMUNITY 2.0

MOLTO DI PIÙ DI UN CMS

Quando si parla di pubblicazione di documenti si entra in un'ottica piuttosto complessa di cui la parte visibile è soltanto quella più appariscente ma anche probabilmente quella meno complessa. Chi autorizza alla pubblicazione di un documento? Quante persone possono lavorare alle bozze? Come vengono gestite le revisioni? Quali tipi di documenti si possono condividere? Rispondere ad alcune di queste domande è già più complicato di quanto non sembri. Alfresco non solo è un CMS ma è un completo sistema di gestione della documentazione che unisce insieme potenza e flessibilità. E' sviluppato in java con la tecnica delle JSP per cui ha probabilmente un complessità leggermente superiore ai classici CMS che abbondano in rete, tuttavia anche le possibilità offerte dal software sono notevoli, per cui è ovvio che il sistema necessiti di un background adeguato. Da provare

ALGORITMI DI MOLTIPLICAZIONE

LA FONDAMENTALE OPERAZIONE DELLA MOLTIPLICAZIONE PUÒ ESSERE COMPIUTA IN MOLTI MODI, ALCUNI ERANO GIÀ NOTI ALLE ANTICHE CIVILTÀ COME QUELLA EGIZIA. VEDIAMO QUALI SONO I METODI CHE VELOCIZZANO LA CPU

La moltiplicazione riveste un ruolo fondamentale proprio perché si tratta di un'operazione molto usata e che si ritrova di frequente nelle più svariate relazioni e formule. Il problema come vedremo era già molto sentito dai nostri antenati che avevano prodotto dei metodi geniali per semplificare l'operazione o per evitare di ricordare a memoria le tabelle. Sta di fatto che esistono modi davvero curiosi per affrontare alcune semplici operazioni aritmetiche che svelano ancora una volta il fascino celato dietro i semplici numeri e le loro strutture di base. Con l'avvento dell'era digitale sono sorte altre esigenze che afferiscono da un lato alla rappresentazione binaria dei numeri e dall'altro alle tematiche dell'efficienza nella realizzazione di operazioni che siano svolte dalle macchine. Così, sono stati introdotti nuovi metodi per affrontare tali problematiche. Esploreremo diversi metodi: da quelli per così dire storici dal fascino intramontabile ai moderni usati nel mondo dell'informatica. Chi è interessato ai soli algoritmi sviluppati per l'algebra al computer dopo aver letto i primi due paragrafi, può direttamente accedere alla seconda parte a partire dalla moltiplicazione di Karatsuba.

MOLTIPLICAZIONE SCOLASTICA

Si tratta del metodo che tutti conosciamo e che indicheremo come scolastico proprio perché usualmente insegnato nei primi anni di studi, lo introduciamo per comprendere l'ambito di studio.

Semplicemente si allineano i due numeri da moltiplicare su due righe e si moltiplica a partire da destra la prima cifra del secondo numero per tutte le cifre del primo, sempre partendo da destra, e si scrive il risultato. In particolare, se il risultato non è un'unica cifra, si scrive la cifra meno significativa, la più significativa si usa come resto e si somma al risultato della moltiplicazione successiva. Si continua così per tutte le altre cifre a seguire del secondo numero. I risultati si riportano uno sotto l'altro ogni volta shiftando di uno a destra. Al termine si fa la somma che indicherà il prodotto. Facciamo un esempio, credetemi non voglio calpestare la sensibilità di alcuno, ma nell'epoca delle calcolatrici e dei computer, per molti questi metodi sono davvero un lontano ricordo:

```
   39106 x
    4785 =
  ----------
   195530
  312848
 273742
156424
---------------
18122210
```

ESEMPIO 1: "Moltiplicazione 4785 x 39106 con la moltiplicazione lunga"

Tale metodo, conosciuto anche come moltiplicazione lunga, ha un significativo valore didattico poiché prevede una serie di passaggi che devono essere eseguiti con precisione per arrivare al risultato finale. Inoltre, è richiesta la conoscenza della cosiddetta "tabellina" ossia tutte le moltiplicazioni elementari tra numeri ad una cifra.

Ad ogni modo ha una certa complessità che lo rende ostico per gli studenti che non siano ordinati e che non conoscano la tabellina. Nel caso in cui si abbia a che fare con numeri binari la moltiplicazione si riduce ad un semplice and logico. Se i numeri di cui si deve fare il prodotto sono composti da n cifre il prodotto comporterà all'incirca un numero di operazioni elementari pari al quadrato di n. Quindi la complessità in definitiva è O(n2).

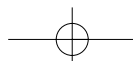
REQUISITI

Conoscenze richieste: *Basi di programmazione*

Software: -

Impegno

Tempo di realizzazione

110-114 29-06-2007 17:11 Pagina 111

ALGORITMO A SPAZIO LOGARITMICO

È molto interessante un'analisi della complessità spaziale in termini di bit. Sia n il numero di bit che rappresenta i due numeri. L'algoritmo che realizza la moltiplicazione lunga necessita di uno spazio proporzionale al logaritmo di n. L'algoritmo può essere facilmente descritto da una sommatoria. Si addizionano le colonne da destra a sinistra, tenendo traccia del resto che viene prodotto di volta in volta. Siano a e b i due numeri da moltiplicare e sia r il risultato. Possiamo estrarre il valore di un bit del risultato, il generico ri come la somma tra il riporto c e la sommatoria dei prodotti dei bit interessati.

$$r_i = c + \sum_{j+k=i} a_j b_k$$

Si parte con i=0. Gli indici j e k vengono rappresentati con O(logn) bit. Un ragionamento induttivo, che non affrontiamo per semplicità, indica che il riporto c non può superare mai n e che il generico ri non può essere maggiore di 2n. Ovviamente, per la prima colonna c sarà zero, si tratta del valore iniziale. Per tutte le altre colonne ci sono al massimo n bit più l'eventuale riporto. Ecco come si può realizzare la funzione in C++.

```
Void multiplica(int a[n],int  b[n]) // vettori di bit
                                        da destra a sinistra
{   x = 0
    for (i=0; i<=2*n-1; i++)
    {   for (j=0; j<=i; j++)
        {   k = i – j;
            x = x + (a[j] _ b[k])
        }
        r[i] = x % 2  // resto della divisione tra x e 2
        x = x/2       // parte intera della divisione tra
                                                    x e 2
    }
}
```

del termine algoritmo, dove si cita un metodo indiano di moltiplicazione con l'aiuto di celle. Anche nel tredicesimo secolo nel libro di matematica del Magreb si fa riferimento al metodo. In Europa è Fibonacci nel 1202 a descriverlo, mentre Leonardo tratta il metodo come un'operazione mentale facilitata nei calcoli intermedi dalla sua scrittura mancina da destra verso sinistra. Gli ossi o bastoncini di Nepero furono usati proprio per realizzare questo metodo. Ma vediamo come funziona.01

	4	7	8	5	
01	1 / 2	2 / 1	2 / 4	1 / 5	3
07	3 / 6	6 / 3	7 / 2	4 / 5	9
15	0 / 4	0 / 7	0 / 8	0 / 5	1
19	0 / 0	0 / 0	0 / 0	0 / 0	0
20	2 / 4	4 / 2	4 / 8	3 / 0	6
	21	11	11	00	

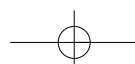
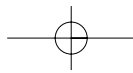
```
007
0015
00019
000020
0000021
00000011
000000011
0000000000
-----------------
187122210
```

ESEMPIO 2: "Moltiplicazione 4785 x 39106 con il crivello moltiplicativo"

CRIVELLO MOLTIPLICATIVO

Facciamo un passo indietro. Esaminiamo alcuni metodi proposti in passato, ancora validi, che forniscono utili riflessioni. Si tratta di un algoritmo equivalente al precedente; qui è usata una griglia, o crivello, che guida le operazioni. Cosicché, la moltiplicazione diventa ancora più facile. Questo metodo ha un interessante ed intenso percorso storico. Il riferimento più antico si trova su un trattato di matematica arabo di Al-Khwarizmi che ricordiamo per essere la radice

Bisogna costruire una tabella posizionando sulla prima riga e sull'ultima colonna i due numeri da moltiplicare. Si guardi l'esempio 2 in cui vengono moltiplicati i due numeri 4785 e 86106. All'incrocio tra le singole cifre si riporta il risultato posizionando le due cifre, la più e la meno significativa, rispettivamente sopra e sotto un separatore diagonale. La presenza di questi separatori costituisce delle pseudo colonne diagonali. Ai margini della tabella vengono riportate le somme di tutte queste diagonali. L'ultimo passo è sommare, tenendo conto ovvia-

110-114 29-06-2007 17:11 Pagina 112

mente dell'eventuale resto, i numeri ottenuti dalle somme diagonali shiftando ogni volta di una cifra a destra. Come si può notare nell'esempio il risultato, il numero 187122210, come ci aspettavamo è uguale al risultato ottenuto con il metodo della moltiplicazione lunga dell'esempio precedente.

NEPERO

Colui che è conosciuto in Italia con il nome di Nepero è John Napier (1550 – 1617) uno studioso scozzese. Non un matematico di professione ma un ricco terriero con l'hobby della matematica. Ad egli si deve l'introduzione di importantissimi concetti matematici come: il logaritmo naturale e di conseguenza il numero e conosciuto come numero di Nepero, la virgola come separatore dei decimali di cui fu un sostenitore, e degli ossi per facilitare le moltiplicazioni oggetto della presente trattazione. Le suo opere sono: Mirifici logarithmorum canonis descriptio che tradotto è "Descrizione della regola meravigliosa dei logaritmi" e Rabdologiae in cui descrive l'uso degli strumenti da lui ideati per velocizzare l'operazione di moltiplicazione.

BASTONCINI DI NEPERO

Il metodo prima visto è ingegnoso e bello. Esso fa uso dei bastoncini di Nepero che per la loro natura reale vengono usati un po' diversamente. I bastoncini di Nepero sono delle astine lunghe e strette che realizzano una generica colonna della tabella del crivello vista. Quindi si possono giustapporre diverse di queste astine. Così si possono rapidamente conoscere i risultati delle moltiplicazioni tra il numero letto nell'intestazione delle astine affiancate per un qualsiasi numero, ad una cifra, basta sommare le diagonali opportune estraendo la riga corrispondente. Ad esempio si voglia moltiplicare 187 x 4; come mostra la **figura 1** bisogna affiancare i tre ossi di Nepero corrispondenti alle cifre 1, 8 e 7 e poi considerare la riga 4. Considerando le diagonali si ottiene 07, 04 e 08, che sommati danno il risultato sperato 748. Nel caso specifico data la presenza dello zero come prima cifra significativa di tutti i numeri è sufficiente affiancarli.
Per cui il crivello non è altro che un'estensione e una generalizzazione del metodo più elementare proposto da Nepero.

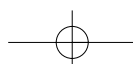
BASTONCINI DI LUCAS-GENAILLE

Un'ingegnosa idea di due francesi, E Lucas che la pensò e H Genaille che la mise in pratica, nel 1885 ha migliorato l'idea dei bastoncini di Nepero. Con questi nuovi bastoncini, più elaborati, si evita di dover trattare il problema del riporto.

Fig. 1: "Moltiplicazione con l'uso dei bastoncini di Nepero"

LA MOLTIPLICAZIONE DEL CONTADINO

Un metodo diffuso in ambienti non scolarizzati è la moltiplicazione binaria o anche detta del contadino. Attraverso questo procedimento si arriva al risultato conoscendo poche operazioni elementari, bastano la moltiplicazione, la divisione intera per 2 e la somma. Non è richiesta la conoscenza della tabellina. Il metodo non è adatto a numeri di grandi dimensioni. Vediamo come funziona sviluppando un esempio. Supponiamo di voler moltiplicare 20 x 3. Si costruiscono due colonne contenenti i due operandi. Il primo numero ad ogni riga viene diviso per due e si estrae la parte intera, per intenderci si tratta dell'operatore pascal div 2. Nella seconda colonna ad ogni riga il numero viene raddoppiato. Si sbarrano i numeri della seconda colonna corrispondenti a numeri pari della prima. I rimanenti vengono sommati e si ottiene così il risultato.
È interessante notare che questo metodo funziona anche scambiando i due operandi nelle intestazioni. Quindi dividendo per fasi successive il numero più piccolo e moltiplicando il più grande. In generale in questo modo si ottengo-

no meno fasi, meno righe, ma si devono trattare numeri di dimensioni più grandi. Questo metodo si può attuare anche con l'uso di oggetti come pietre o carte; sistemati sempre su due colonne e opportunamente divisi e moltiplicati per due con il procedimento descritto.

20	3
10	6
5	12
2	24
1	48

	60

ESEMPIO 3: "Moltiplicazione 20 x 3 con la moltiplicazione del contadino"

ANTICHE GRIGLIEO

Il reperto riportato di seguito è stato ritrovato da un matematico persiano del 15° secolo. Nella tabella si calcola il prodotto di due numeri in formato sessagesimale.

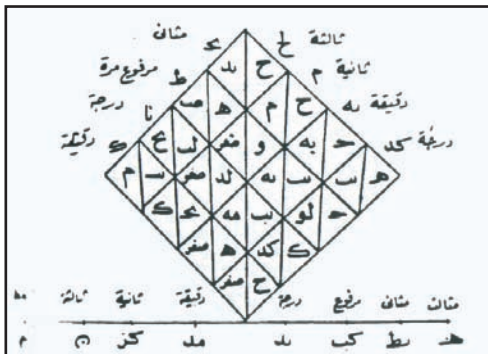



FIGURA 3: "Moltiplicazione con uso di celle triangolari"

Questo reperto è estratto da "A history o agorithms" di J.L Chabert e AL. che a loro volta lo hanno reperito da Miftah al-hisab (The key of calculation) A.S. Damirdash e M.H. al-Hafni, 1967

MOLTIPLICAZIONE KARATSUBA

Dopo una piacevole quanto intensa immersione in metodi del passato ritorniamo all'era digitale. I problemi più sensibili riguardano il trattamento di numeri di grandi dimensioni. Sappiamo ad esempio che usualmente i linguaggi di programmazione trattano i numeri interi con dimensioni "relativamente piccole". Ad esempio, l'integer pascal o l'int C++ sono tipi interi rappresentati con due byte e quindi sono definiti nell'intervallo [-32768,32767]. Esistono tipi che prevedono un maggior numero di byte. Comunque si possono presentare problemi con numeri molto grandi che normalmente non si possono trattare. Del resto la moltiplicazione scolastica può non essere efficiente in termini di prestazioni. Una linea guida che risolve le problematiche esposte fa riferimento alla moltiplicazione Karatsuba. Questa si fonda su una semplice quanto potente proprietà algebrica. Consideriamo due numeri a e b entrambi li scomponiamo in termini ossia esprimiamo sia a che b come la somma di due numeri, ovviamente più piccoli. Cosicché possiamo scrivere:

$$a=a0 + a1$$
$$b=b0 + b1$$

Sarà più facile trattare con i nuovi numeri introdotti: a0, a1, b0 e b1; essendo più piccoli. Il prodotto che vogliamo calcolare, si può scrivere semplicemente sfruttando le basilari proprietà algebriche dei binomi:

$$P = a * b$$
$$= (a0 + a1) * (b0 + b1)$$
$$= a0*b0 + (a0b12 + a1*b0) + a1*b1$$

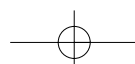
Quindi diventa la somma di tre termini. È interessante la generalizzazione del metodo in cui a e b sono costituiti da bit (nel caso specifico solo due) di un numero binario che possiamo esprimere in una qualsiasi base w. Allora si possono riscrivere le formule appena elaborate tenendo conto della base.

$$a=a0 + a1*w$$
$$b=b0 + b1*w$$
$$P = a * b$$
$$= (a0 + a1*w) * (b0 + b1*w)$$
$$= a0*b0 + (a0*b1 + a1*b0)w + a1*b1*w^2$$

In questo caso si evidenziano i tre termini riferiti alle posizioni di esponente 0, 1 e 2 rispetto alla base w. Tale relazione si può scrivere:

$$P=P1+P2w+P3w^2$$

Con i termini P corrispondenti alle aggregazioni di a0, a1, b0 e b1 presenti nella relazione precedente. Per concludere il metodo va detto che non si hanno limitazioni a soli numeri binari con soli due bit. Se ad esempio i bit sono quattro:

$$a = a0+a1*w+a2*w^2+a3*w^3$$

il numero può essere riscritto come due bit in base w^2, basta raggruppare.

a = (a0+a1w) + (a2+23w)*w^2

La moltiplicazione Karatsuba abbatte la complessità dell'algoritmo. Si passa da $O(n^2)$ della moltiplicazione lunga a $O(n^{\log 3})$, ossia $O(n^{1,58})$ ottenuto ricorsivamente, che è sensibilmente più efficiente. La differenza emerge man mano che aumenta il numero di bit. In **figura 2** è riportata un confronto tra i due metodi.

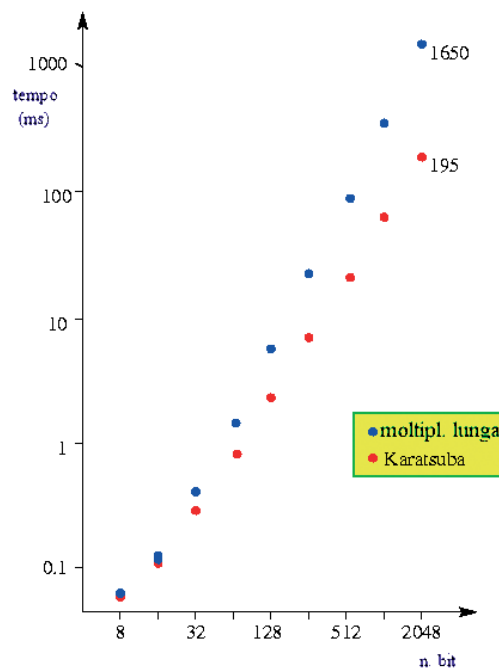



Fig. 2: "Confronto tra le prestazioni dei due metodi moltiplicazione lunga e karatsuba""

METODI A SCOMPOSIZIONI MULTIPLE

Un altro metodo sulla scia di Karatsuba per la moltiplicazione tra interi che ha una complessità meno che quadratica è opera di Toom, con successiva formalizzazione e aggiustamento di Cook ed è conosciuto anche come Toom3. Oggi è ampiamente usato per la moltiplicazioni di grandi polinomi. L'idea alla base è la stessa di Karatsuba ma rispetto ad essa è evoluta poiché prevede la partizione non solo in due parti bensì in k. Non si produrranno, quindi come nel caso precedente, dei binomi ma polinomi. Se k = 3 si ottiene il citato Toom3, con cui in letteratura spesso si identifica in generale il metodo Toom Cook; in realtà non è così Toom 3 è solo un'istanza del metodo più generale (k=3). Tale algoritmo è leggermente più efficiente del precedente, si ha infatti $O(c(k)\, n^z)$, dove z = log(2k-1) / log(k). c è un valore che non cresce in modo apprezzabile con k, di fatto tiene conto di addizioni e interpolazioni. Il fattore z ci indica che all'aumentare di k le cose migliorano. Ad esempio Toom 3 ha complessità $O(n^{1,465})$.

L'algoritmo più efficiente per la moltiplicazione di interi è la trasformata di Fourier. Si parte sempre dalla idea di scomporre il numero come una sommatoria, nel caso particolare di w bit seguendo appunto il conosciuto metodo di riduzione della complessità della trasformata di Fourier

$$a = \sum_{i=0}^{m} 2^{wi} a_i$$

$$b = \sum_{j=0}^{m} 2^{wj} b_j$$

Possiamo così esprimere il prodotto tra i due numeri come:

$$ab = \sum_{i=0}^{m} \sum_{j=0}^{m} 2^{w(i+j)} a_i b_j = \sum_{k=0}^{2m} 2^{wk} \sum_{i=0}^{m} a_i b_{k-i} = \sum_{k=0}^{2m} 2^{wk} c_k$$

In definitiva questo metodo consta di 4 stadi:

1. Si calcolano le FFT (fast Fourier trasforms) di ai e bj
2. Si moltiplicano i due risultati volta per volta
3. Si calcola l'inversa della trasformata di Fourier
4. Si sommano le parti di ck (resto), che sono più grandi di 2^w, con il termine ck+1

Si ottiene un brillante risultato in termini di complessità computazionale: O(n logn); che al momento è il migliore.

CONCLUSIONI

Penso che ancora una volta sia stato rimarcato il concetto di algoritmo che non è associato in modo sterile alla presenza dell'elaboratore ma è soprattutto uno strumento nelle mani di tutti per risolvere i problemi con gli strumenti che si hanno a disposizione. Se il contadino aveva solo pietre per poter far di conto il metodo da lui ideato per moltiplicare ha un valore inestimabile anche perché ci conferma ancora una volta che per fare qualcosa esistono molti modi. I metodi per computer sono più complessi e prevedono la scomposizione del numero. Insomma ce né per tutti i gusti. Spero che vi siate divertiti come è capitato a me nell'affrontare questa ricerca. Alla prossima!

Fabio Grimaldi